

autogluon_each_location

October 18, 2023

```
[1]: # config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = ["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
use_test_data = True
tune_and_test_length = 0.5 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val_
    ↪ and score_test in stack models.

sample_weight = None #'sample_weight' #None
weight_evaluation = False
sample_weight_estimated = 1

run_analysis = True

[2]: import pandas as pd
import numpy as np
```

```

import warnings
warnings.filterwarnings("ignore")

def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,
    # we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")

    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so
    # get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1
            hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
    columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]

    date_calc = None
    if "date_calc" in X.columns:

```

```

        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X

def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column
    # with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    X = feature_engineering(X)

    return X

def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")

    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated

    y_train['ds'] = pd.to_datetime(y_train['time'])

```

```

y_train.drop(columns=['time'], inplace=True)
y_train.sort_values(by='ds', inplace=True)
y_train.set_index('ds', inplace=True)

return X_train_observed, X_train_estimated, X_test, y_train

def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,
location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =
handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -
pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =
X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

```

```

X_train = pd.merge(X_train, y_train, how="inner", left_index=True,
↳right_index=True)

# print number of nans in y
print(f"Number of nans in y: {X_train['y'].isna().sum()}")

X_train["location"] = location
X_test["location"] = location

return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed = pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,
↳X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)

```

```

Processing location A...
COUNT1 29667
COUNT2 1
index: 2019-06-02 22:00:00
index AFTER: 2019-06-02 22:00:00

```

```

COUNT1 4392
COUNT2 2
index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 0
Processing location B...
COUNT1 29232
COUNT2 1
index: 2019-01-01 00:00:00
index AFTER: 2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 4
Processing location C...
COUNT1 29206
COUNT2 1
index: 2019-01-01 00:00:00
index AFTER: 2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index: 2022-10-28 22:00:00
index AFTER: 2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index: 2023-05-01 00:00:00
index AFTER: 2023-05-01 00:00:00
Number of nans in y: 6059

```

1 Feature engineering

```

[3]: import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],
               inplace=True)

```

```

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())

if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique() # Assuming 'location' is the name
    ↪ of the column representing locations

    grouped_dfs = [] # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),
    ↪ repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())

to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",
    ↪ "dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",
    ↪ "wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",
    ↪ "rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
    ↪ gm3", "air_density_2m:kgm3"]

```

```

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)

```

```

absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00      7.700      1.22825
2019-06-02 23:00:00      7.700      1.22350
2019-06-03 00:00:00      7.875      1.21975
2019-06-03 01:00:00      8.425      1.21800
2019-06-03 02:00:00      8.950      1.21800

```

```

ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00      1728.949951      0.000000
2019-06-02 23:00:00      1689.824951      0.000000
2019-06-03 00:00:00      1563.224976      0.000000
2019-06-03 01:00:00      1283.425049      6546.899902
2019-06-03 02:00:00      1003.500000      102225.898438

```

```

clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00      0.00      1728.949951      0.0
2019-06-02 23:00:00      0.00      1689.824951      0.0
2019-06-03 00:00:00      0.00      1563.224976      0.0
2019-06-03 01:00:00      0.75      1283.425049      0.0
2019-06-03 02:00:00      23.10      1003.500000      0.0

```

```

dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  ...  \
ds
2019-06-02 22:00:00      280.299988      0.000      0.000000  ...
2019-06-02 23:00:00      280.299988      0.000      0.000000  ...
2019-06-03 00:00:00      280.649994      0.000      0.000000  ...
2019-06-03 01:00:00      281.674988      0.300      7743.299805  ...
2019-06-03 02:00:00      282.500000      11.975      60137.601562  ...

```

```

visibility:m  wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 22:00:00      40386.476562      3.600      -3.575
2019-06-02 23:00:00      33770.648438      3.350      -3.350
2019-06-03 00:00:00      13595.500000      3.050      -2.950
2019-06-03 01:00:00      2321.850098      2.725      -2.600
2019-06-03 02:00:00      11634.799805      2.550      -2.350

```

```

wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \

```


ds			
2019-06-02 22:00:00	-0.500		0.0
2019-06-02 23:00:00	0.275		0.0
2019-06-03 00:00:00	0.750		0.0
2019-06-03 01:00:00	0.875		0.0
2019-06-03 02:00:00	0.925		0.0

	is_estimated	y	location	hour	year
ds					
2019-06-02 22:00:00	0	0.00	A	22	2019
2019-06-02 23:00:00	0	0.00	A	23	2019
2019-06-03 00:00:00	0	0.00	A	0	2019
2019-06-03 01:00:00	0	0.00	A	1	2019
2019-06-03 02:00:00	0	19.36	A	2	2019

[5 rows x 50 columns]

```
[4]: def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /
        loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
import numpy as np

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the
    bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output
        dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
```

```

        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
        ↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

        # Shuffle and split the remaining data
        remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
        remaining_size1 = int(len(remaining_data) * frac1)

        output_data1 = pd.concat([output_data1, remaining_data.iloc[:
        ↪remaining_size1]])
        output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
        ↪]])

    return output_data1, output_data2

```

```

[5]: from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:

```

```

#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())

# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
    ↳Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
test_set = TabularDataset(data[data["ds"] >= split_time])

# shuffle test_set and only grab tune_and_test_length percent of it, rest goes
    ↳to train_set
test_set, new_train_set = split_and_shuffle_data(test_set, 40,
    ↳tune_and_test_length)

print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))
print("Length of test set", len(test_set))

if use_groups:
    test_set = test_set.drop(columns=['group'])

tuning_data = None
if use_tune_data:
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data =
↳split_and_shuffle_data(loc_test_set, 40, 0.5)
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↳shape[0], tuning_data.shape[0] + test_data.shape[0])

```

```

else:
    tuning_data = test_set
    print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in
    ↪ the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)

else:
    if use_test_data:
        test_data = test_set
        print("Shape of test", test_data.shape[0])

train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of
↪ rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)

train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)

```

Length of train set before adding test set 82026

Length of train set after adding test set 87486

Length of test set 5459

Shapes of tuning and test 2728 2731 5459

```

[6]: if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,
    ↪ label="y", sample=None)

```

train_data dataset summary

	count	unique	top	freq	\
ceiling_height_agl:m	72280	59980			
clear_sky_energy_1h:J	87486	46359			
clear_sky_rad:W	87486	19918			

cloud_base_agl:m	81454	61360		
diffuse_rad:W	87486	11092		
diffuse_rad_1h:J	87486	46319		
direct_rad:W	87486	14181		
direct_rad_1h:J	87486	40118		
ds	87486	36794	2021-02-05 14:00:00	3
effective_cloud_cover:p	87486	5655		
elevation:m	87486	3		
fresh_snow_1h:cm	87486	39		
fresh_snow_3h:cm	87486	70		
fresh_snow_6h:cm	87486	96		
hour	87486	24		
is_day:idx	87486	5		
is_estimated	87486	2		
is_in_shadow:idx	87486	5		
location	87486	3		A 31872
msl_pressure:hPa	87486	3708		
precip_type_5min:idx	87486	15		
pressure_100m:hPa	87486	3730		
pressure_50m:hPa	87486	3775		
relative_humidity_1000hPa:p	87486	3799		
sfc_pressure:hPa	87486	3795		
snow_depth:cm	87486	487		
snow_water:kgm2	87486	161		
sun_azimuth:d	87486	83179		
sun_elevation:d	87486	72262		
super_cooled_liquid_water:kgm2	87486	53		
t_1000hPa:K	87486	1989		
total_cloud_cover:p	87486	5556		
visibility:m	87486	85949		
wind_speed_10m:ms	87486	596		
y	87486	11321		
year	87486	5		

	first	last	mean \
ceiling_height_agl:m	NaT	NaT	2861.929806
clear_sky_energy_1h:J	NaT	NaT	530297.395771
clear_sky_rad:W	NaT	NaT	147.308425
cloud_base_agl:m	NaT	NaT	1740.241802
diffuse_rad:W	NaT	NaT	40.267497
diffuse_rad_1h:J	NaT	NaT	145328.6257
direct_rad:W	NaT	NaT	51.524847
direct_rad_1h:J	NaT	NaT	185338.05854
ds	2019-01-01	2023-04-30 22:00:00	
effective_cloud_cover:p	NaT	NaT	67.052836
elevation:m	NaT	NaT	11.414718
fresh_snow_1h:cm	NaT	NaT	0.008783
fresh_snow_3h:cm	NaT	NaT	0.026713

fresh_snow_6h:cm	NaT	NaT	0.05322
hour	NaT	NaT	11.499589
is_day:idx	NaT	NaT	0.490147
is_estimated	NaT	NaT	0.06241
is_in_shadow:idx	NaT	NaT	0.556952
location	NaT	NaT	
msl_pressure:hPa	NaT	NaT	1009.434307
precip_type_5min:idx	NaT	NaT	0.084976
pressure_100m:hPa	NaT	NaT	995.759729
pressure_50m:hPa	NaT	NaT	1001.884211
relative_humidity_1000hPa:p	NaT	NaT	73.779918
sfc_pressure:hPa	NaT	NaT	1008.035963
snow_depth:cm	NaT	NaT	0.197574
snow_water:kgm2	NaT	NaT	0.090839
sun_azimuth:d	NaT	NaT	179.584247
sun_elevation:d	NaT	NaT	-0.705998
super_cooled_liquid_water:kgm2	NaT	NaT	0.058256
t_1000hPa:K	NaT	NaT	279.675551
total_cloud_cover:p	NaT	NaT	73.72398
visibility:m	NaT	NaT	32944.238197
wind_speed_10m:ms	NaT	NaT	3.032943
y	NaT	NaT	294.447861
year	NaT	NaT	2020.568365

	std	min	25%	50%	\
ceiling_height_agl:m	2532.377528	27.8	1082.3125	1856.075	
clear_sky_energy_1h:J	831839.646633	0.0	0.0	9084.9	
clear_sky_rad:W	231.107565	0.0	0.0	2.325	
cloud_base_agl:m	1808.519208	27.5	598.15625	1174.775	
diffuse_rad:W	61.119566	0.0	0.0	1.35	
diffuse_rad_1h:J	218036.296903	0.0	0.0	12531.2	
direct_rad:W	114.236728	0.0	0.0	0.0	
direct_rad_1h:J	406379.39471	0.0	0.0	0.0	
ds					
effective_cloud_cover:p	34.132847	0.0	42.125	79.7	
elevation:m	7.881545	6.0	6.0	7.0	
fresh_snow_1h:cm	0.110515	0.0	0.0	0.0	
fresh_snow_3h:cm	0.277575	0.0	0.0	0.0	
fresh_snow_6h:cm	0.474579	0.0	0.0	0.0	
hour	6.920464	0.0	5.0	12.0	
is_day:idx	0.486133	0.0	0.0	0.5	
is_estimated	0.2419	0.0	0.0	0.0	
is_in_shadow:idx	0.484138	0.0	0.0	1.0	
location					
msl_pressure:hPa	12.993994	944.375	1001.45	1010.325	
precip_type_5min:idx	0.32995	0.0	0.0	0.0	
pressure_100m:hPa	12.922092	929.975	987.85	996.75	
pressure_50m:hPa	12.979336	935.75	993.925012	1002.85	

relative_humidity_1000hPa:p	14.200631	19.575	64.4	76.15
sfc_pressure:hPa	13.038723	941.55	1000.05	1009.0
snow_depth:cm	1.28439	0.0	0.0	0.0
snow_water:kgm2	0.240248	0.0	0.0	0.0
sun_azimuth:d	97.419022	6.983	94.4135	180.00663
sun_elevation:d	24.006117	-49.932	-17.969875	-0.453875
super_cooled_liquid_water:kgm2	0.106959	0.0	0.0	0.0
t_1000hPa:K	6.551665	258.025	275.1	278.975
total_cloud_cover:p	33.851299	0.0	53.475	92.825
visibility:m	17949.64428	132.375	16564.5125	36910.75
wind_speed_10m:ms	1.758713	0.025	1.675	2.7
y	774.531815	-0.0	0.0	0.0
year	1.102625	2019.0	2020.0	2021.0

	75%	max	dtypes \
ceiling_height_agl:m	3916.78125	12285.775	float64
clear_sky_energy_1h:J	831169.675	3006697.2	float64
clear_sky_rad:W	231.34375	835.65	float64
cloud_base_agl:m	2080.39375	11673.725	float64
diffuse_rad:W	67.15	334.75	float64
diffuse_rad_1h:J	243365.275	1182265.4	float64
direct_rad:W	32.225	683.4	float64
direct_rad_1h:J	122454.125	2445897.0	float64
ds			datetime64[ns]
effective_cloud_cover:p	98.5	100.0	float64
elevation:m	24.0	24.0	float64
fresh_snow_1h:cm	0.0	7.1	float64
fresh_snow_3h:cm	0.0	20.6	float64
fresh_snow_6h:cm	0.0	34.0	float64
hour	17.0	23.0	int64
is_day:idx	1.0	1.0	float64
is_estimated	0.0	1.0	int64
is_in_shadow:idx	1.0	1.0	float64
location			object
msl_pressure:hPa	1018.42505	1044.1	float64
precip_type_5min:idx	0.0	5.0	float64
pressure_100m:hPa	1004.8	1030.875	float64
pressure_50m:hPa	1010.92505	1037.25	float64
relative_humidity_1000hPa:p	85.175	100.0	float64
sfc_pressure:hPa	1017.1	1043.725	float64
snow_depth:cm	0.0	18.2	float64
snow_water:kgm2	0.1	5.65	float64
sun_azimuth:d	264.601138	348.48752	float64
sun_elevation:d	16.004499	49.94375	float64
super_cooled_liquid_water:kgm2	0.1	1.375	float64
t_1000hPa:K	284.225	303.25	float64
total_cloud_cover:p	99.9	100.0	float64
visibility:m	48289.05	75326.58	float64

wind_speed_10m:ms	4.05	13.275	float64
y	183.7125	5733.42	float64
year	2021.0	2023.0	int64

	missing_count	missing_ratio	raw_type \
ceiling_height_agl:m	15206	0.173811	float
clear_sky_energy_1h:J			float
clear_sky_rad:W			float
cloud_base_agl:m	6032	0.068948	float
diffuse_rad:W			float
diffuse_rad_1h:J			float
direct_rad:W			float
direct_rad_1h:J			float
ds			datetime
effective_cloud_cover:p			float
elevation:m			float
fresh_snow_1h:cm			float
fresh_snow_3h:cm			float
fresh_snow_6h:cm			float
hour			int
is_day:idx			float
is_estimated			int
is_in_shadow:idx			float
location			object
msl_pressure:hPa			float
precip_type_5min:idx			float
pressure_100m:hPa			float
pressure_50m:hPa			float
relative_humidity_1000hPa:p			float
sfc_pressure:hPa			float
snow_depth:cm			float
snow_water:kgm2			float
sun_azimuth:d			float
sun_elevation:d			float
super_cooled_liquid_water:kgm2			float
t_1000hPa:K			float
total_cloud_cover:p			float
visibility:m			float
wind_speed_10m:ms			float
y			float
year			int

	variable_type	special_types
ceiling_height_agl:m	numeric	
clear_sky_energy_1h:J	numeric	
clear_sky_rad:W	numeric	
cloud_base_agl:m	numeric	
diffuse_rad:W	numeric	

diffuse_rad_1h:J	numeric
direct_rad:W	numeric
direct_rad_1h:J	numeric
ds	
effective_cloud_cover:p	numeric
elevation:m	category
fresh_snow_1h:cm	numeric
fresh_snow_3h:cm	numeric
fresh_snow_6h:cm	numeric
hour	numeric
is_day:idx	category
is_estimated	category
is_in_shadow:idx	category
location	category
msl_pressure:hPa	numeric
precip_type_5min:idx	category
pressure_100m:hPa	numeric
pressure_50m:hPa	numeric
relative_humidity_1000hPa:p	numeric
sfc_pressure:hPa	numeric
snow_depth:cm	numeric
snow_water:kgm2	numeric
sun_azimuth:d	numeric
sun_elevation:d	numeric
super_cooled_liquid_water:kgm2	numeric
t_1000hPa:K	numeric
total_cloud_cover:p	numeric
visibility:m	numeric
wind_speed_10m:ms	numeric
y	numeric
year	category

test_data dataset summary

	count	unique	top	freq	\
ceiling_height_agl:m	2100	2065			
clear_sky_energy_1h:J	2731	1138			
clear_sky_rad:W	2731	997			
cloud_base_agl:m	2374	2332			
diffuse_rad:W	2731	983			
diffuse_rad_1h:J	2731	1138			
direct_rad:W	2731	750			
direct_rad_1h:J	2731	932			
ds	2731	2200	2023-04-10 19:00:00	3	
effective_cloud_cover:p	2731	1348			
elevation:m	2731	3			
fresh_snow_1h:cm	2731	19			
fresh_snow_3h:cm	2731	36			
fresh_snow_6h:cm	2731	50			

hour	2731	24	
is_day:idx	2731	5	
is_estimated	2731	1	
is_in_shadow:idx	2731	5	
location	2731	3	A 1094
msl_pressure:hPa	2731	1525	
precip_type_5min:idx	2731	10	
pressure_100m:hPa	2731	1536	
pressure_50m:hPa	2731	1557	
relative_humidity_1000hPa:p	2731	1523	
sfc_pressure:hPa	2731	1575	
snow_depth:cm	2731	61	
snow_water:kgm2	2731	61	
sun_azimuth:d	2731	2716	
sun_elevation:d	2731	2652	
super_cooled_liquid_water:kgm2	2731	29	
t_1000hPa:K	2731	774	
total_cloud_cover:p	2731	1126	
visibility:m	2731	2729	
wind_speed_10m:ms	2731	366	
y	2731	895	
year	2731	2	

	first	last \
ceiling_height_agl:m	NaT	NaT
clear_sky_energy_1h:J	NaT	NaT
clear_sky_rad:W	NaT	NaT
cloud_base_agl:m	NaT	NaT
diffuse_rad:W	NaT	NaT
diffuse_rad_1h:J	NaT	NaT
direct_rad:W	NaT	NaT
direct_rad_1h:J	NaT	NaT
ds	2022-10-28 22:00:00	2023-04-30 19:00:00
effective_cloud_cover:p	NaT	NaT
elevation:m	NaT	NaT
fresh_snow_1h:cm	NaT	NaT
fresh_snow_3h:cm	NaT	NaT
fresh_snow_6h:cm	NaT	NaT
hour	NaT	NaT
is_day:idx	NaT	NaT
is_estimated	NaT	NaT
is_in_shadow:idx	NaT	NaT
location	NaT	NaT
msl_pressure:hPa	NaT	NaT
precip_type_5min:idx	NaT	NaT
pressure_100m:hPa	NaT	NaT
pressure_50m:hPa	NaT	NaT
relative_humidity_1000hPa:p	NaT	NaT

sfc_pressure:hPa	NaT	NaT
snow_depth:cm	NaT	NaT
snow_water:kgm2	NaT	NaT
sun_azimuth:d	NaT	NaT
sun_elevation:d	NaT	NaT
super_cooled_liquid_water:kgm2	NaT	NaT
t_1000hPa:K	NaT	NaT
total_cloud_cover:p	NaT	NaT
visibility:m	NaT	NaT
wind_speed_10m:ms	NaT	NaT
y	NaT	NaT
year	NaT	NaT

	mean	std	min \
ceiling_height_agl:m	3361.682133	2562.862274	28.0
clear_sky_energy_1h:J	285528.142658	573252.521662	0.0
clear_sky_rad:W	79.243464	159.124874	0.0
cloud_base_agl:m	1685.111501	1833.56975	27.5
diffuse_rad:W	26.230813	48.360421	0.0
diffuse_rad_1h:J	94649.301538	172723.786046	0.0
direct_rad:W	32.798068	92.244541	0.0
direct_rad_1h:J	118054.008202	329601.815692	0.0
ds			
effective_cloud_cover:p	66.798654	36.717964	0.0
elevation:m	11.193336	7.806119	6.0
fresh_snow_1h:cm	0.023984	0.147555	0.0
fresh_snow_3h:cm	0.069498	0.352141	0.0
fresh_snow_6h:cm	0.13885	0.57722	0.0
hour	11.544855	6.879849	0.0
is_day:idx	0.378341	0.472171	0.0
is_estimated	1.0	0.0	1.0
is_in_shadow:idx	0.677133	0.452911	0.0
location			
msl_pressure:hPa	1010.736333	14.355902	972.15
precip_type_5min:idx	0.076254	0.344931	0.0
pressure_100m:hPa	996.906224	14.168772	959.19995
pressure_50m:hPa	1003.137999	14.243079	965.15
relative_humidity_1000hPa:p	71.631472	14.652551	21.7
sfc_pressure:hPa	1009.397775	14.318453	971.15
snow_depth:cm	0.119965	0.56196	0.0
snow_water:kgm2	0.080511	0.19473	0.0
sun_azimuth:d	180.975998	94.222121	14.914
sun_elevation:d	-8.945472	22.095926	-49.887
super_cooled_liquid_water:kgm2	0.035088	0.082444	0.0
t_1000hPa:K	275.52987	4.271781	259.975
total_cloud_cover:p	72.32056	37.085445	0.0
visibility:m	34504.948017	17242.154257	270.3
wind_speed_10m:ms	3.109676	1.782531	0.125

y	179.379421	641.546947	0.0	
year	2022.693153	0.46127	2022.0	
	25%	50%	75%	max \
ceiling_height_agl:m	1245.55625	2784.275	4919.18125	12294.9
clear_sky_energy_1h:J	0.0	0.0	220909.2	2551917.2
clear_sky_rad:W	0.0	0.0	62.7625	709.825
cloud_base_agl:m	516.7625	1000.8	2066.9875	10813.7
diffuse_rad:W	0.0	0.0	32.0	280.5
diffuse_rad_1h:J	0.0	0.0	116208.0	986147.0
direct_rad:W	0.0	0.0	4.7625	511.7
direct_rad_1h:J	0.0	0.0	24326.65	1844204.9
ds				
effective_cloud_cover:p	36.3125	83.05	99.825	100.0
elevation:m	6.0	7.0	24.0	24.0
fresh_snow_1h:cm	0.0	0.0	0.0	2.3
fresh_snow_3h:cm	0.0	0.0	0.0	4.8
fresh_snow_6h:cm	0.0	0.0	0.0	6.3
hour	6.0	12.0	17.5	23.0
is_day:idx	0.0	0.0	1.0	1.0
is_estimated	1.0	1.0	1.0	1.0
is_in_shadow:idx	0.0	1.0	1.0	1.0
location				
msl_pressure:hPa	1000.35	1010.6	1021.4625	1041.2
precip_type_5min:idx	0.0	0.0	0.0	3.0
pressure_100m:hPa	986.8	996.95	1007.675	1027.8
pressure_50m:hPa	992.9	1003.175	1014.0	1034.1
relative_humidity_1000hPa:p	61.775	73.55	82.9625	99.775
sfc_pressure:hPa	999.1	1009.5	1020.275	1040.6
snow_depth:cm	0.0	0.0	0.0	4.9
snow_water:kgm2	0.0	0.0	0.1	2.15
sun_azimuth:d	101.047372	179.89075	260.65412	347.81226
sun_elevation:d	-26.72725	-8.178	6.393625	41.09175
super_cooled_liquid_water:kgm2	0.0	0.0	0.0	0.75
t_1000hPa:K	272.6625	275.45	278.5	285.825
total_cloud_cover:p	44.5875	96.775	100.0	100.0
visibility:m	20902.55	36141.85	48867.949	73937.67
wind_speed_10m:ms	1.6	2.8	4.2375	9.9
y	0.0	0.0	40.397706	5043.72
year	2022.0	2023.0	2023.0	2023.0

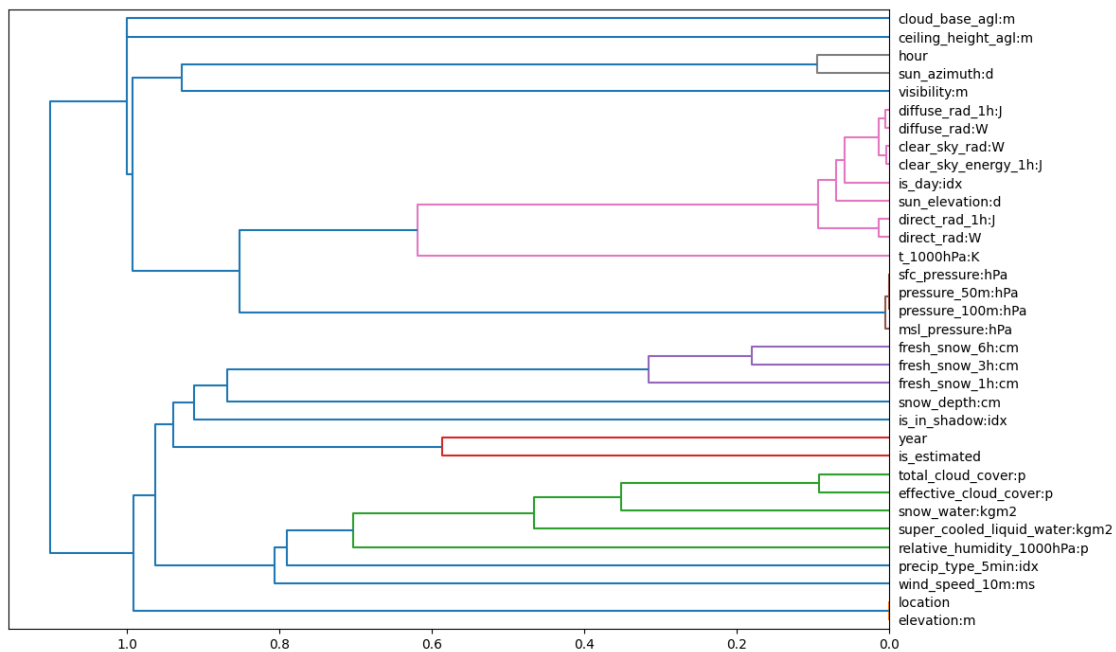
	dtypes	missing_count	missing_ratio	\
ceiling_height_agl:m	float64	631	0.231051	
clear_sky_energy_1h:J	float64			
clear_sky_rad:W	float64			
cloud_base_agl:m	float64	357	0.130721	
diffuse_rad:W	float64			
diffuse_rad_1h:J	float64			

direct_rad:W	float64
direct_rad_1h:J	float64
ds	datetime64[ns]
effective_cloud_cover:p	float64
elevation:m	float64
fresh_snow_1h:cm	float64
fresh_snow_3h:cm	float64
fresh_snow_6h:cm	float64
hour	int64
is_day:idx	float64
is_estimated	int64
is_in_shadow:idx	float64
location	object
msl_pressure:hPa	float64
precip_type_5min:idx	float64
pressure_100m:hPa	float64
pressure_50m:hPa	float64
relative_humidity_1000hPa:p	float64
sfc_pressure:hPa	float64
snow_depth:cm	float64
snow_water:kgm2	float64
sun_azimuth:d	float64
sun_elevation:d	float64
super_cooled_liquid_water:kgm2	float64
t_1000hPa:K	float64
total_cloud_cover:p	float64
visibility:m	float64
wind_speed_10m:ms	float64
y	float64
year	int64

	raw_type	variable_type	special_types
ceiling_height_agl:m	float	numeric	
clear_sky_energy_1h:J	float	numeric	
clear_sky_rad:W	float	numeric	
cloud_base_agl:m	float	numeric	
diffuse_rad:W	float	numeric	
diffuse_rad_1h:J	float	numeric	
direct_rad:W	float	numeric	
direct_rad_1h:J	float	numeric	
ds	datetime		
effective_cloud_cover:p	float	numeric	
elevation:m	float	category	
fresh_snow_1h:cm	float	category	
fresh_snow_3h:cm	float	numeric	
fresh_snow_6h:cm	float	numeric	
hour	int	numeric	
is_day:idx	float	category	

is_estimated	int	category
is_in_shadow:idx	float	category
location	object	category
msl_pressure:hPa	float	numeric
precip_type_5min:idx	float	category
pressure_100m:hPa	float	numeric
pressure_50m:hPa	float	numeric
relative_humidity_1000hPa:p	float	numeric
sfc_pressure:hPa	float	numeric
snow_depth:cm	float	numeric
snow_water:kgm2	float	numeric
sun_azimuth:d	float	numeric
sun_elevation:d	float	numeric
super_cooled_liquid_water:kgm2	float	numeric
t_1000hPa:K	float	numeric
total_cloud_cover:p	float	numeric
visibility:m	float	numeric
wind_speed_10m:ms	float	numeric
y	float	numeric
year	int	category

1.0.1 Feature Distance



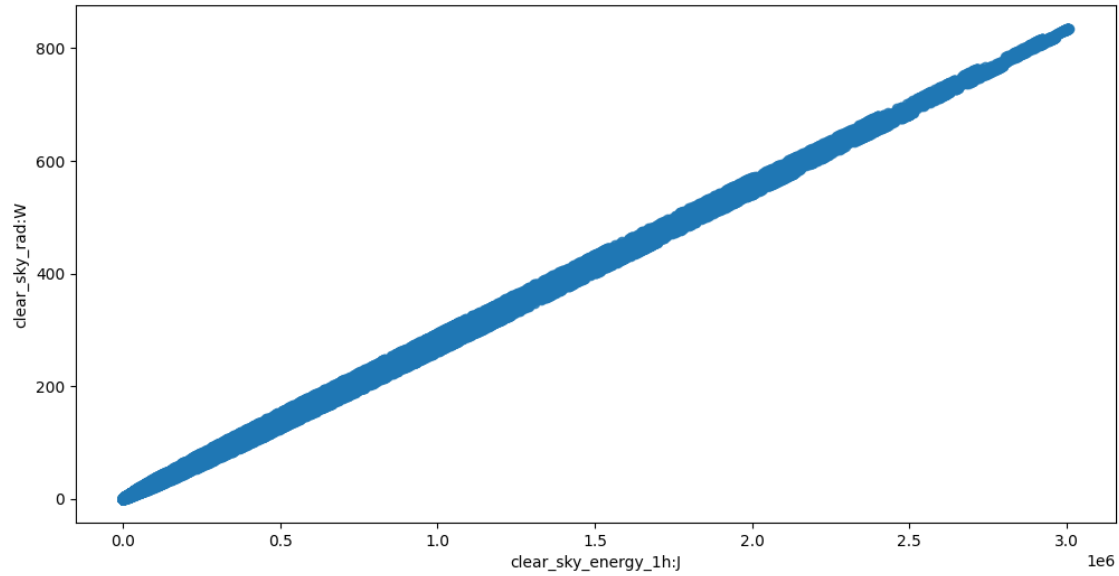
The following feature groups are considered as near-duplicates:

Distance threshold: ≤ 0.01 . Consider keeping only some of the columns within each group:

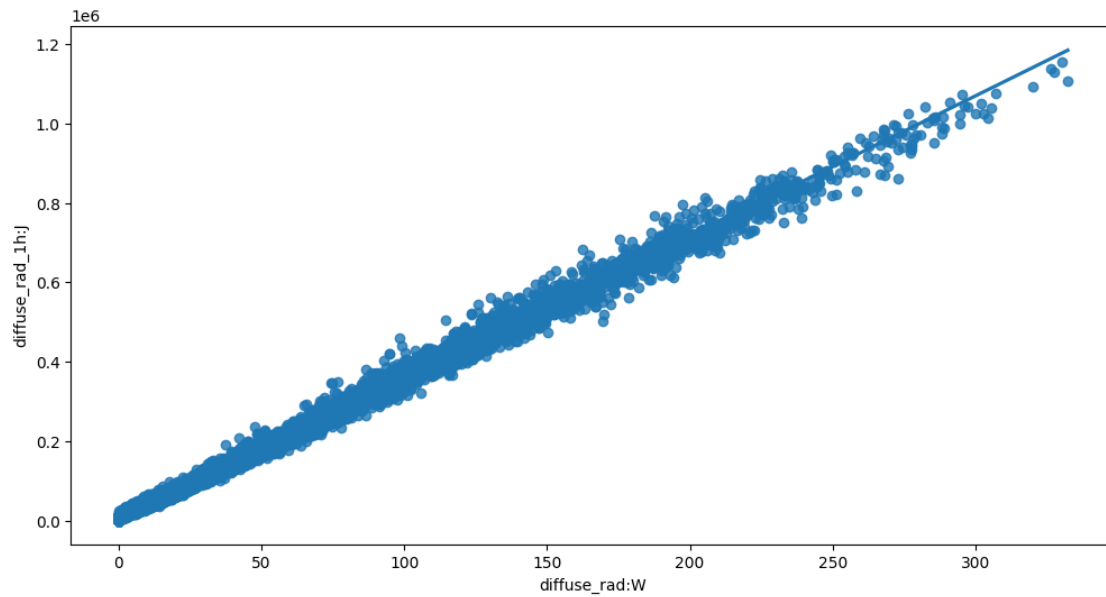
- elevation:m, location - distance 0.00

- clear_sky_energy_1h:J, clear_sky_rad:W - distance 0.00
- diffuse_rad:W, diffuse_rad_1h:J - distance 0.00
- msl_pressure:hPa, pressure_100m:hPa, pressure_50m:hPa, sfc_pressure:hPa - distance 0.00

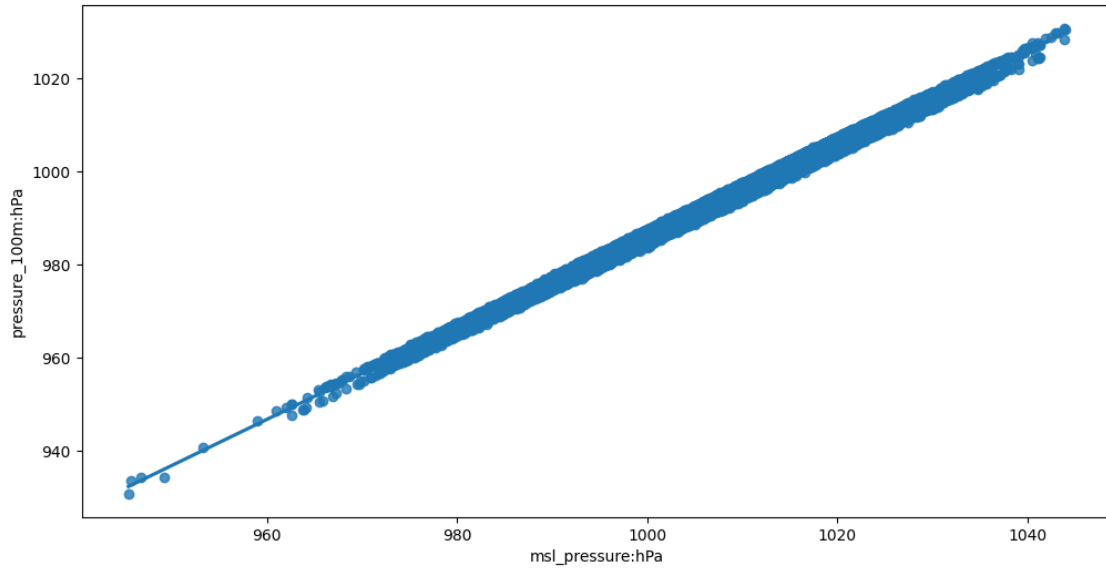
Feature interaction between clear_sky_energy_1h:J/clear_sky_rad:W



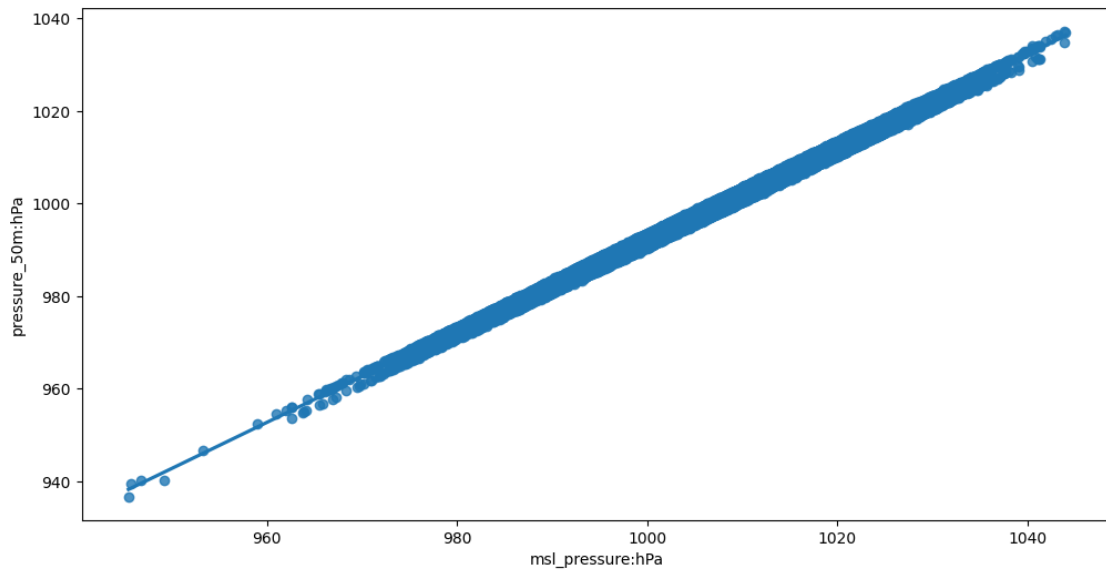
Feature interaction between diffuse_rad:W/diffuse_rad_1h:J



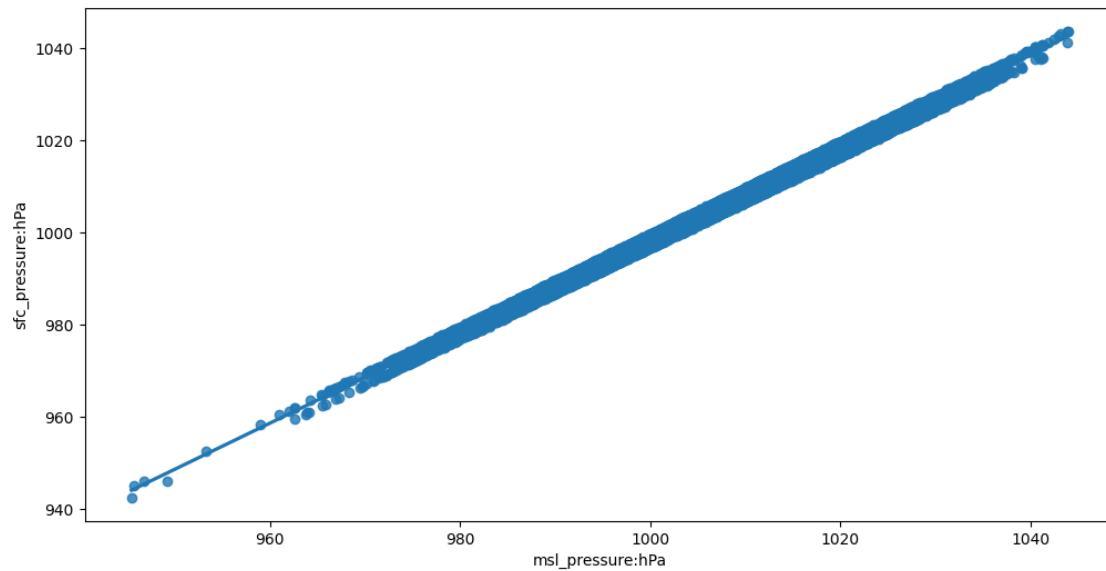
Feature interaction between `msl_pressure:hPa/pressure_100m:hPa`



Feature interaction between `msl_pressure:hPa/pressure_50m:hPa`



Feature interaction between `msl_pressure:hPa/sfc_pressure:hPa`

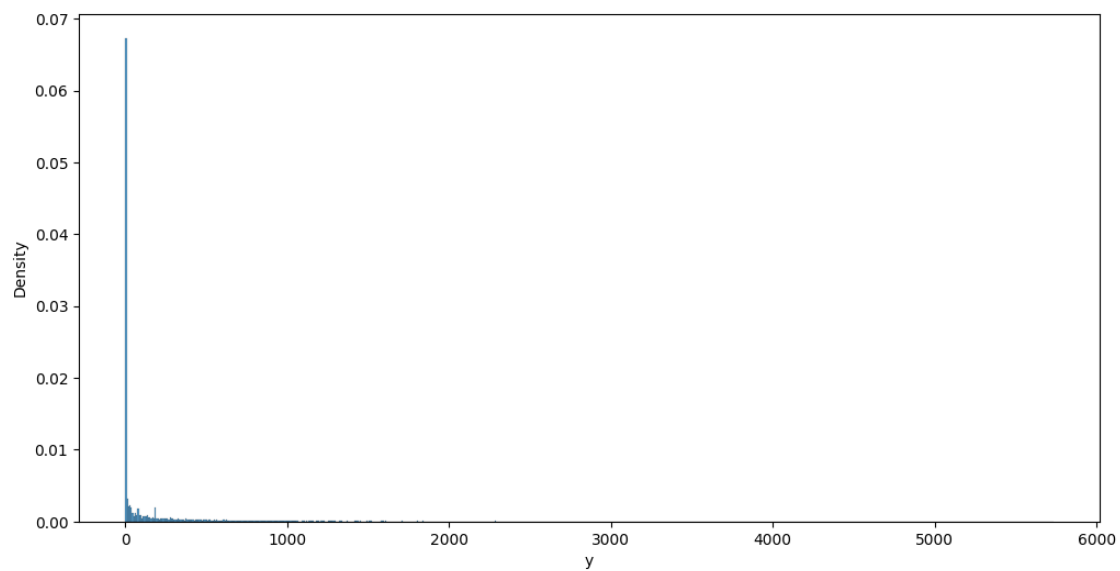


```
[7]: if run_analysis:
      auto.target_analysis(train_data=train_data, label="y", sample=None)
```

1.1 Target variable analysis

	count	mean	std	min	25%	50%	75%	max	dtypes
y	87486	294.447861	774.531815	-0.0	0.0	0.0	183.7125	5733.42	float64

	unique	missing_count	missing_ratio	raw_type	special_types
y	11321			float	

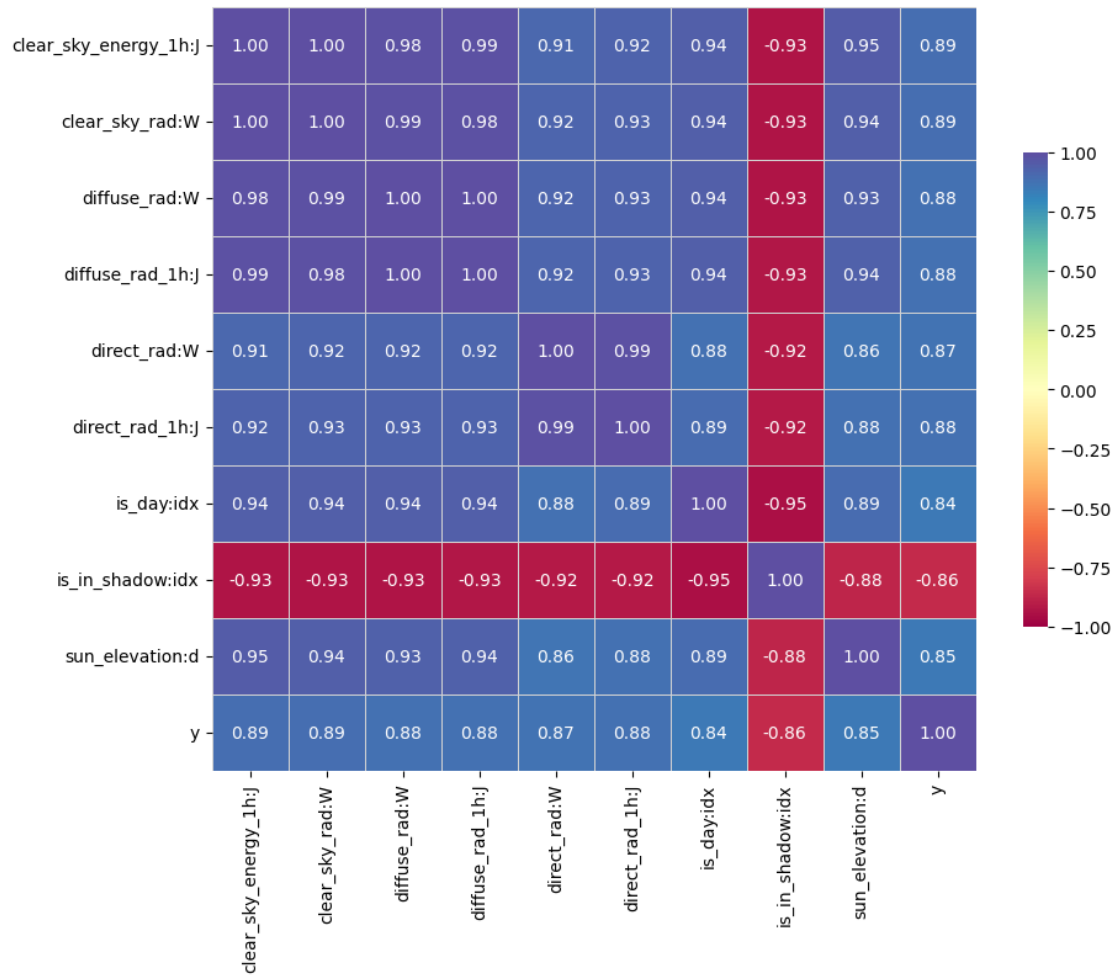


1.1.1 Distribution fits for target variable

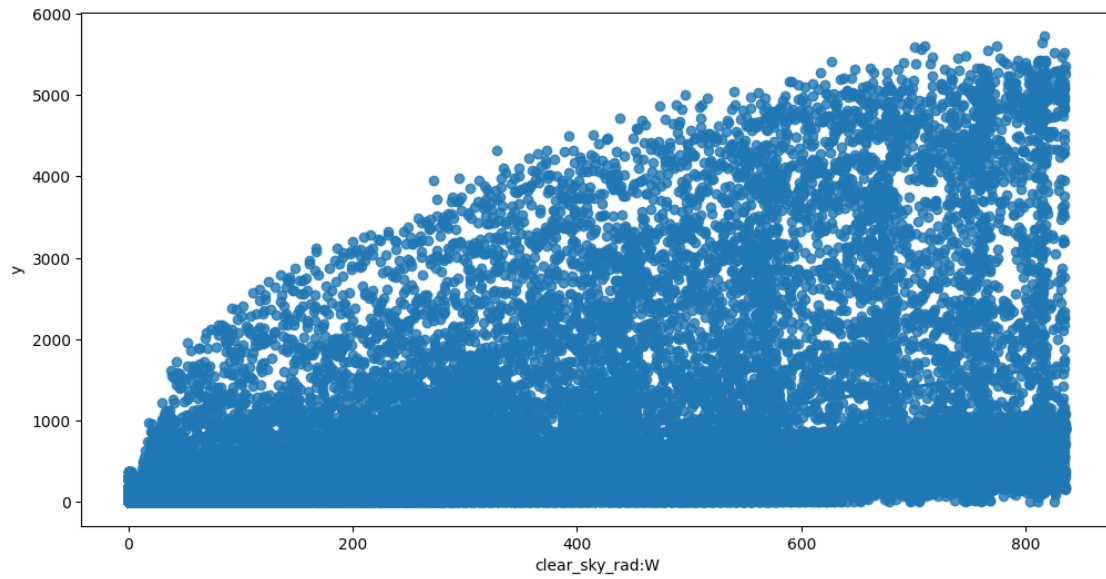
- none of the [attempted](#) distribution fits satisfy specified minimum p-value threshold: 0.01

1.1.2 Target variable correlations

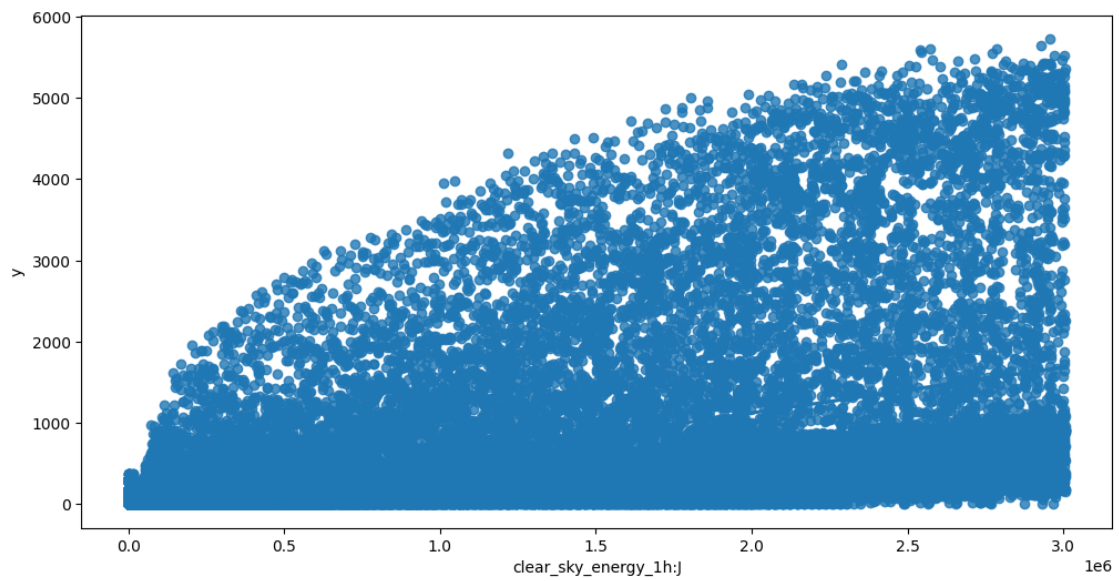
train_data - spearman correlation matrix; focus: absolute correlation for $\gamma \geq 0.5$



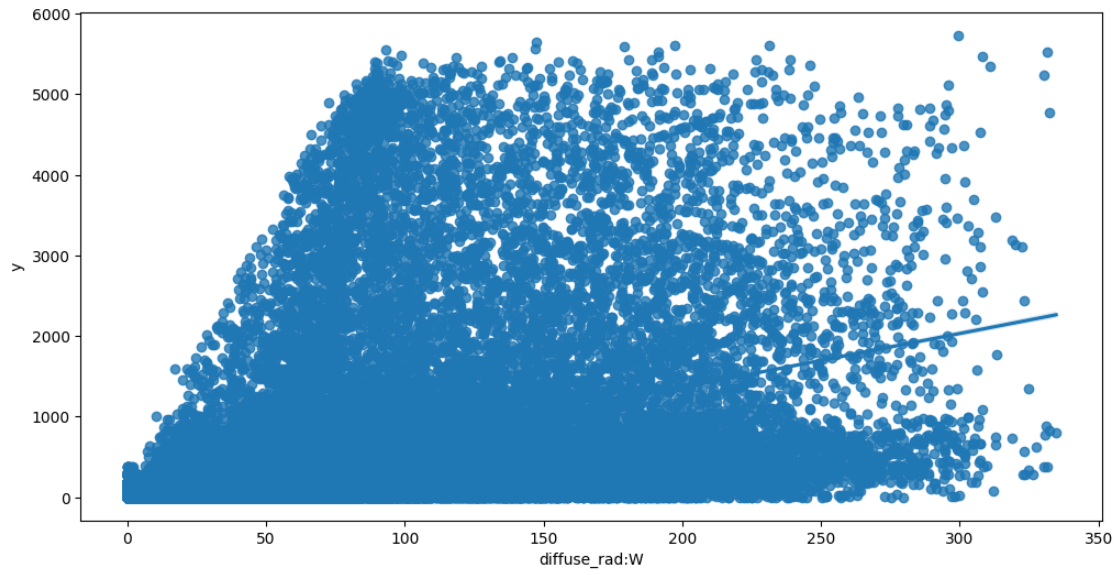
Feature interaction between clear_sky_rad:W/y in train_data



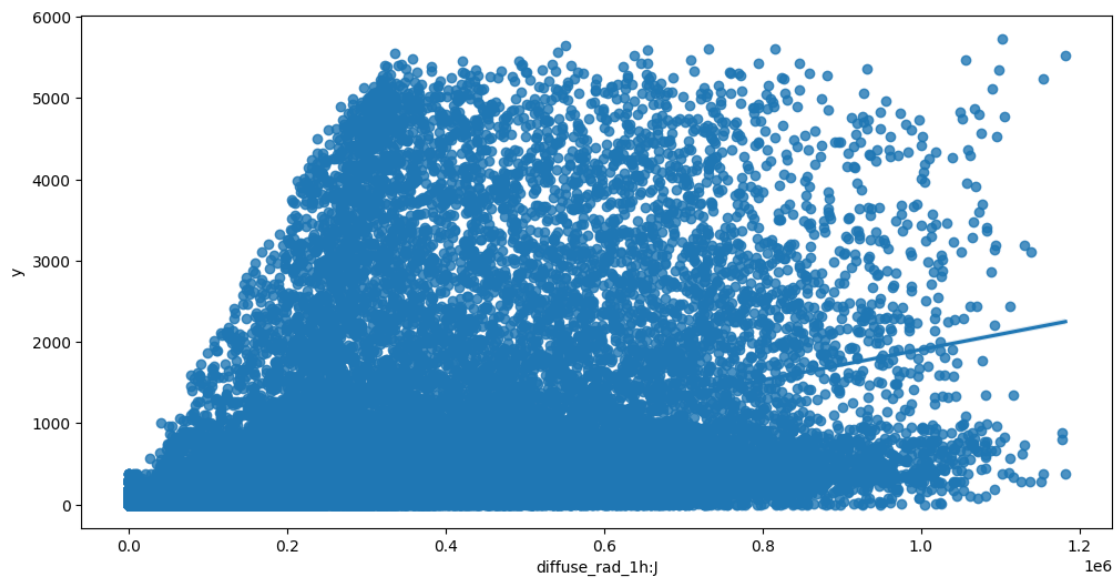
Feature interaction between clear_sky_energy_1h:J/y in train_data



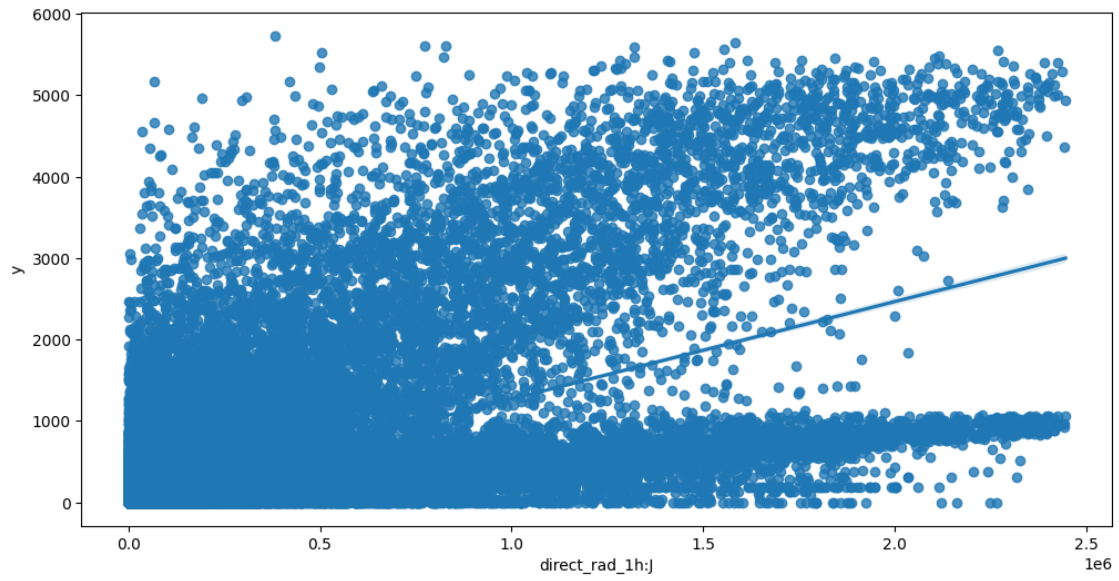
Feature interaction between diffuse_rad:W/y in train_data



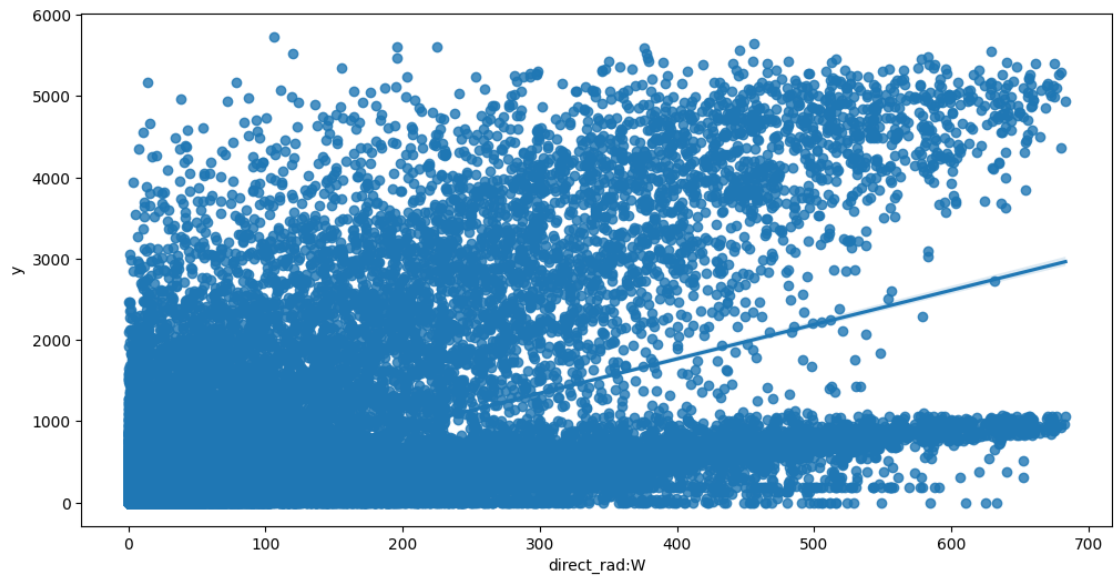
Feature interaction between `diffuse_rad_1h:J/y` in `train_data`



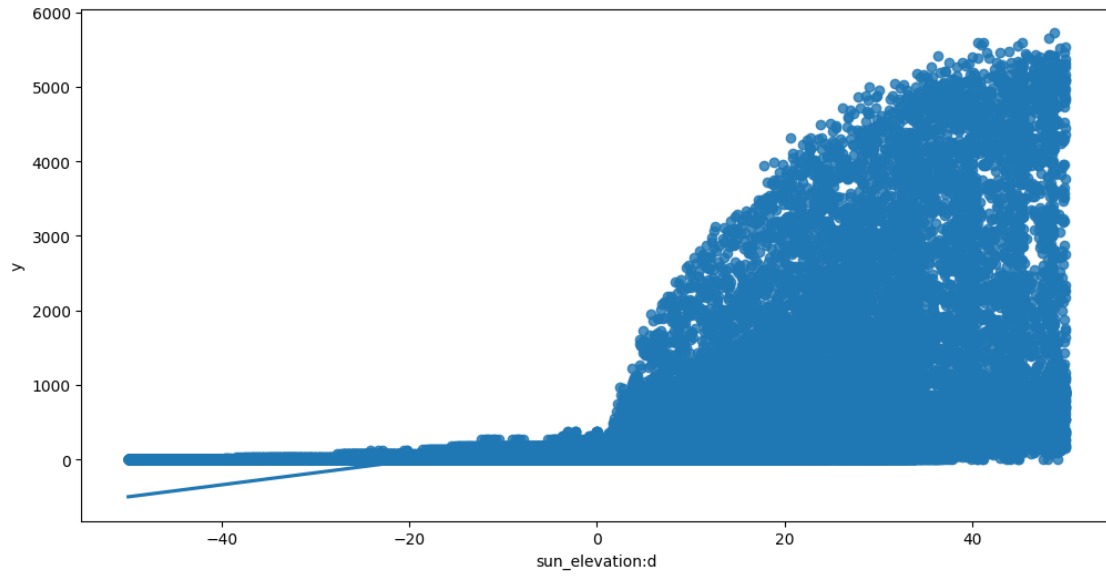
Feature interaction between `direct_rad_1h:J/y` in `train_data`



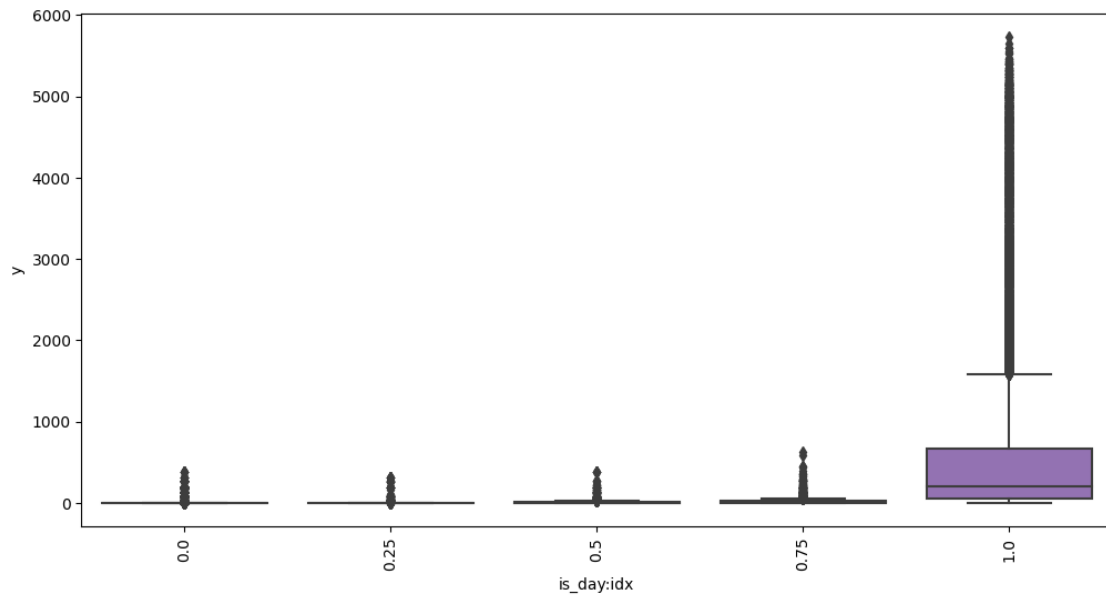
Feature interaction between `direct_rad:W/y` in `train_data`



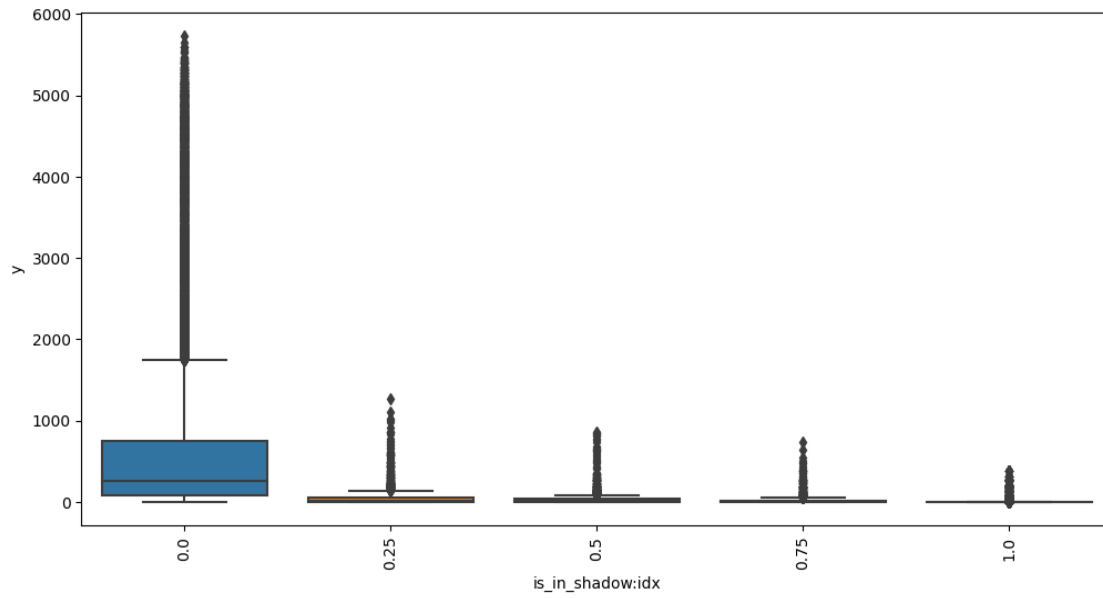
Feature interaction between `sun_elevation:d/y` in `train_data`



Feature interaction between is_day:idx/y in train_data



Feature interaction between is_in_shadow:idx/y in train_data



2 Starting

```
[8]: import os

# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for
    ↪ filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 95
Now creating submission number: 96
New filename: submission_96
```

```
[9]: predictors = [None, None, None]
```

```

[10]: def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both
    ↪ train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",
    ↪ train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]
    ↪ == loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",
    ↪ tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",
    ↪ tuning_data[tuning_data["location"] == loc].shape[0])
            if use_test_data:
                print("Test data sample weight sum:",
    ↪ test_data[test_data["location"] == loc]["sample_weight"].sum())
                print("Test data number of rows:", test_data[test_data["location"]
    ↪ == loc].shape[0])
        predictor = TabularPredictor(
            label=label,
            eval_metric=metric,
            path=f"AutogluonModels/{new_filename}_{loc}",
            # sample_weight=sample_weight,
            # weight_evaluation=weight_evaluation,
            # groups="group" if use_groups else None,
        ).fit(
            train_data=train_data[train_data["location"] == loc].
    ↪ drop(columns=["ds"]),
            time_limit=time_limit,
            # presets=presets,
            num_stack_levels=num_stack_levels,
            num_bag_folds=num_bag_folds if not use_groups else 2, # just put
    ↪ somethin, will be overwritten anyways
            num_bag_sets=num_bag_sets,
            tuning_data=tuning_data[tuning_data["location"] == loc].
    ↪ reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
            use_bag_holdout=use_bag_holdout,
            # holdout_frac=holdout_frac,
        )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc].
    ↪ drop(columns=["sample_weight"])

```



```

        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)

```

```

Beginning AutoGluon training ... Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_96_A/"
AutoGluon Version: 0.8.2
Python Version: 3.10.12
Operating System: Linux
Platform Machine: x86_64
Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail: 213.35 GB / 315.93 GB (67.5%)
Train Data Rows: 31872
Train Data Columns: 34
Tuning Data Rows: 1093
Tuning Data Columns: 34
Label Column: y
Preprocessing data ...
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
    Label info (max, min, mean, stddev): (5733.42, 0.0, 649.68162,
1178.37671)
    If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
    Available Memory: 131870.2 MB
    Train Data (Original) Memory Usage: 10.61 MB (0.0% of available memory)
    Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
    Stage 1 Generators:
        Fitting AsTypeFeatureGenerator...
            Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
    Stage 2 Generators:
        Fitting FillNaFeatureGenerator...
    Stage 3 Generators:
        Fitting IdentityFeatureGenerator...
    Stage 4 Generators:
        Fitting DropUniqueFeatureGenerator...
    Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...

```

Training model for location A...

Useless Original Features (Count: 2): ['elevation:m', 'location']

These features carry no predictive signal and should be manually investigated.

This is typically a feature which has the same value for all rows.

These features do not need to be present at inference time.

Types of features in original data (raw dtype, special dtypes):

```
('float', []) : 29 | ['ceiling_height_agl:m',  
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',  
...]
```

```
('int', []) : 3 | ['is_estimated', 'hour', 'year']
```

Types of features in processed data (raw dtype, special dtypes):

```
('float', []) : 29 | ['ceiling_height_agl:m',  
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',  
...]
```

```
('int', []) : 2 | ['hour', 'year']
```

```
('int', ['bool']) : 1 | ['is_estimated']
```

0.1s = Fit runtime

32 features in original data used to generate 32 features in processed data.

Train Data (Processed) Memory Usage: 8.21 MB (0.0% of available memory)

Data preprocessing and feature engineering runtime = 0.17s ...

AutoGluon will gauge predictive performance using evaluation metric:

'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the metric value.

To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for early stopping).

User-specified model hyperparameters to be fit:

```
{  
    'NN_TORCH': {},  
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],  
    'GBMLarge'],  
    'CAT': {},  
    'XGB': {},  
    'FASTAI': {},  
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',  
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':  
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},  
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',  
'problem_types': ['regression', 'quantile']}}],  
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',  
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':  
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},  
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
```

```

'problem_types': ['regression', 'quantile']}]},
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 1799.83s of
the 1799.83s of remaining time.
    -140.7608      = Validation score    (-mean_absolute_error)
    0.03s         = Training    runtime
    0.38s         = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 1799.11s of
the 1799.11s of remaining time.
    -140.9566      = Validation score    (-mean_absolute_error)
    0.03s         = Training    runtime
    0.4s          = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1798.62s of the
1798.61s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -92.5916      = Validation score    (-mean_absolute_error)
    28.8s         = Training    runtime
    17.06s        = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 1759.94s of the
1759.94s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -95.843       = Validation score    (-mean_absolute_error)
    24.6s         = Training    runtime
    7.22s         = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 1731.24s of
the 1731.23s of remaining time.
    -108.2786     = Validation score    (-mean_absolute_error)
    8.28s         = Training    runtime
    1.2s          = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 1720.54s of the
1720.54s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -103.3862     = Validation score    (-mean_absolute_error)
    189.07s       = Training    runtime
    0.09s         = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 1530.31s of the
1530.3s of remaining time.
    -111.9213     = Validation score    (-mean_absolute_error)
    1.74s         = Training    runtime
    1.2s          = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 1526.15s of
the 1526.15s of remaining time.

```

```

    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -107.2384      = Validation score    (-mean_absolute_error)
    38.19s        = Training    runtime
    0.47s         = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 1486.22s of the
1486.21s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -101.7946      = Validation score    (-mean_absolute_error)
    13.08s         = Training    runtime
    0.51s          = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 1470.88s of
the 1470.88s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -86.9395       = Validation score    (-mean_absolute_error)
    112.67s        = Training    runtime
    0.3s           = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 1356.81s of the
1356.81s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -94.8567       = Validation score    (-mean_absolute_error)
    94.53s         = Training    runtime
    26.81s         = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1250.29s of the
1250.29s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -92.6033       = Validation score    (-mean_absolute_error)
    58.16s         = Training    runtime
    39.56s         = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 1213.46s of the
1213.46s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -95.7537       = Validation score    (-mean_absolute_error)
    48.63s         = Training    runtime
    11.28s         = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 1185.16s of the
1185.16s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -103.2615      = Validation score    (-mean_absolute_error)
    380.47s        = Training    runtime
    0.18s          = Validation runtime

```

```

Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 992.38s of
the 992.38s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -107.9135      = Validation score    (-mean_absolute_error)
    76.8s         = Training runtime
    0.93s         = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 951.37s of the
951.36s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -101.4482     = Validation score    (-mean_absolute_error)
    23.55s        = Training runtime
    0.89s         = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 939.29s of the
939.29s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -86.5488      = Validation score    (-mean_absolute_error)
    229.71s       = Training runtime
    0.63s         = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 820.72s of the
820.72s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -94.2604      = Validation score    (-mean_absolute_error)
    189.14s       = Training runtime
    47.42s        = Validation runtime
Completed 2/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
711.91s of remaining time.
    -85.6831      = Validation score    (-mean_absolute_error)
    0.42s         = Training runtime
    0.0s          = Validation runtime
AutoGluon training complete, total runtime = 1088.53s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_96_A/")
Evaluation: mean_absolute_error on test data: -91.38656916356734
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -91.38656916356734,
    "root_mean_squared_error": -314.3477051569919,
    "mean_squared_error": -98814.47973746709,
    "r2": 0.8946273600380232,
    "pearsonr": 0.9469860621779503,

```

```

    "median_absolute_error": -1.3219637870788574
}

```

Evaluation on test data:
-91.38656916356734

```

[11]: import matplotlib.pyplot as plt

leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] == loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,
↳test_data[test_data["location"] == loc]["y"])
        if use_tune_data:
            plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,
↳tuning_data[tuning_data["location"] == loc]["y"])
            plt.show()

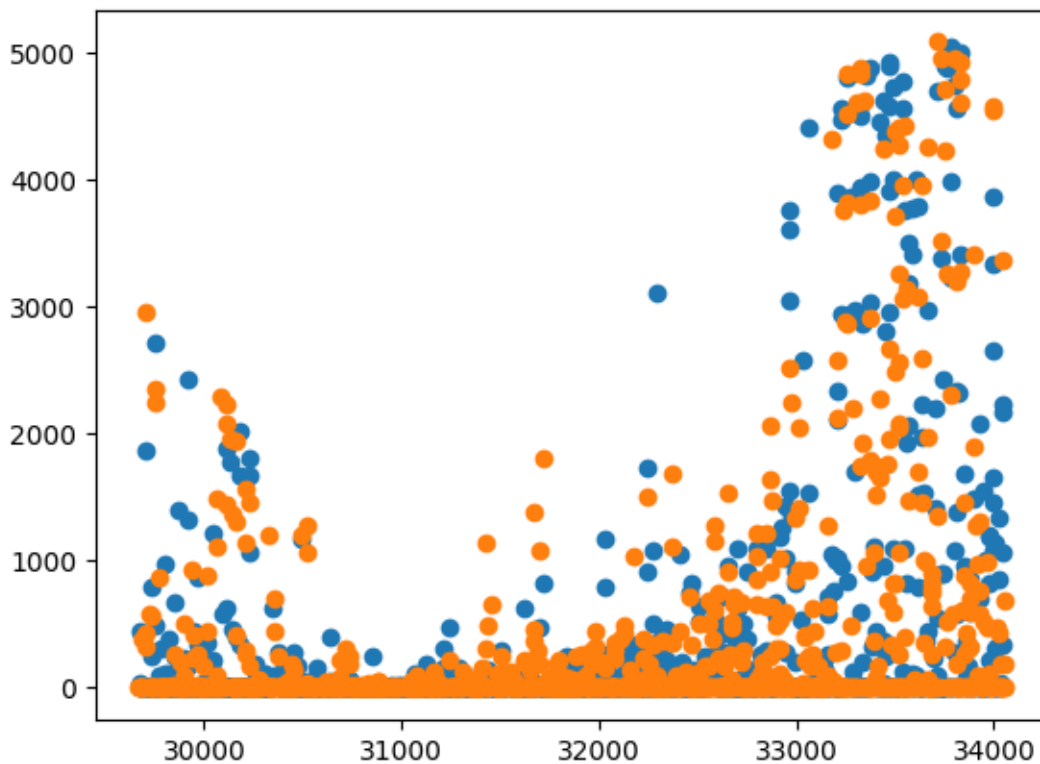
        return lb
    else:
        return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)

```

	model	score_test	score_val	pred_time_test
pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal	
fit_time_marginal	stack_level	can_infer	fit_order	
0	WeightedEnsemble_L2	-91.386569	-85.683123	3.000079
40.194988	288.281092		0.003309	0.000674
0.418749	2	True	12	
1	LightGBMXT_BAG_L1	-93.124416	-92.603287	2.613239
39.563026	58.155759		2.613239	39.563026
58.155759	1	True	3	
2	LightGBMLarge_BAG_L1	-94.537043	-94.260380	7.344643
47.419793	189.143018		7.344643	47.419793
189.143018	1	True	11	
3	NeuralNetTorch_BAG_L1	-94.574389	-86.548840	0.383532
0.631288	229.706584		0.383532	0.631288
229.706584	1	True	10	
4	LightGBM_BAG_L1	-96.718877	-95.753749	1.501362
11.281142	48.632016		1.501362	11.281142
48.632016	1	True	4	
5	CatBoost_BAG_L1	-101.025488	-103.261487	0.141366
0.182645	380.473184		0.141366	0.182645
380.473184	1	True	6	
6	RandomForestMSE_BAG_L1	-101.818436	-108.278639	0.588561

1.204999	8.276278		0.588561	1.204999
8.276278	1	True	5	
7	ExtraTreesMSE_BAG_L1	-103.820493	-111.921296	0.605485
1.198890	1.739662		0.605485	1.198890
1.739662	1	True	7	
8	XGBoost_BAG_L1	-103.905230	-101.448236	0.339590
0.894776	23.548844		0.339590	0.894776
23.548844	1	True	9	
9	NeuralNetFastAI_BAG_L1	-106.185664	-107.913464	1.093756
0.929719	76.798589		1.093756	0.929719
76.798589	1	True	8	
10	KNeighborsDist_BAG_L1	-124.177233	-140.956611	0.019751
0.397748	0.032993		0.019751	0.397748
0.032993	1	True	2	
11	KNeighborsUnif_BAG_L1	-124.918514	-140.760811	0.193960
0.381145	0.031478		0.193960	0.381145
0.031478	1	True	1	



```
[12]: loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

Beginning AutoGluon training ... Time limit = 1800s

```

AutoGluon will save models to "AutogluonModels/submission_96_B/"
AutoGluon Version: 0.8.2
Python Version: 3.10.12
Operating System: Linux
Platform Machine: x86_64
Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail: 209.65 GB / 315.93 GB (66.4%)
Train Data Rows: 31020
Train Data Columns: 34
Tuning Data Rows: 898
Tuning Data Columns: 34
Label Column: y
Preprocessing data ...
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
    Label info (max, min, mean, stddev): (1152.3, -0.0, 99.56591, 196.469)
    If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
    Available Memory: 130041.11 MB
    Train Data (Original) Memory Usage: 10.28 MB (0.0% of available memory)

Training model for location B...

    Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
    Stage 1 Generators:
        Fitting AsTypeFeatureGenerator...
            Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
    Stage 2 Generators:
        Fitting FillNaFeatureGenerator...
    Stage 3 Generators:
        Fitting IdentityFeatureGenerator...
    Stage 4 Generators:
        Fitting DropUniqueFeatureGenerator...
    Stage 5 Generators:
        Fitting DropDuplicatesFeatureGenerator...
    Useless Original Features (Count: 2): ['elevation:m', 'location']
        These features carry no predictive signal and should be manually
investigated.
        This is typically a feature which has the same value for all
rows.
        These features do not need to be present at inference time.
    Types of features in original data (raw dtype, special dtypes):
        ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',

```



```

...]
      ('int', [])      : 3 | ['is_estimated', 'hour', 'year']
Types of features in processed data (raw dtype, special dtypes):
      ('float', [])    : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
...]
      ('int', [])      : 2 | ['hour', 'year']
      ('int', ['bool']) : 1 | ['is_estimated']
0.1s = Fit runtime
32 features in original data used to generate 32 features in processed
data.
Train Data (Processed) Memory Usage: 7.95 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.17s ...
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
  'NN_TORCH': {},
  'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
'GBMLarge'],
  'CAT': {},
  'XGB': {},
  'FASTAI': {},
  'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
  'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
  'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 1799.83s of
the 1799.83s of remaining time.
-23.6782          = Validation score    (-mean_absolute_error)
0.03s            = Training    runtime
0.42s            = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 1799.32s of

```

```

the 1799.32s of remaining time.
    -23.6489          = Validation score    (-mean_absolute_error)
    0.03s            = Training    runtime
    0.41s            = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1798.8s of the
1798.8s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -14.4523          = Validation score    (-mean_absolute_error)
    28.56s            = Training    runtime
    18.37s            = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 1764.38s of the
1764.37s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -14.2832          = Validation score    (-mean_absolute_error)
    30.02s            = Training    runtime
    18.38s            = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 1728.8s of
the 1728.8s of remaining time.
    -14.8681          = Validation score    (-mean_absolute_error)
    8.88s             = Training    runtime
    1.07s             = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 1717.84s of the
1717.84s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -15.9903          = Validation score    (-mean_absolute_error)
    192.97s           = Training    runtime
    0.11s             = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 1523.59s of the
1523.59s of remaining time.
    -13.8493          = Validation score    (-mean_absolute_error)
    1.56s             = Training    runtime
    1.06s             = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 1519.9s of
the 1519.9s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -13.9749          = Validation score    (-mean_absolute_error)
    37.9s             = Training    runtime
    0.48s             = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 1480.24s of the
1480.24s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -14.852           = Validation score    (-mean_absolute_error)
    84.55s            = Training    runtime

```

24.94s = Validation runtime

Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 1390.49s of the 1390.49s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy

-11.6634 = Validation score (-mean_absolute_error)

141.09s = Training runtime

0.31s = Validation runtime

Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 1248.01s of the 1248.0s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy

-13.492 = Validation score (-mean_absolute_error)

94.06s = Training runtime

21.93s = Validation runtime

Repeating k-fold bagging: 2/20

Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1144.13s of the 1144.13s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with ParallelLocalFoldFittingStrategy

-14.2934 = Validation score (-mean_absolute_error)

58.21s = Training runtime

34.22s = Validation runtime

Fitting model: LightGBM_BAG_L1 ... Training model for up to 1107.92s of the 1107.92s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with ParallelLocalFoldFittingStrategy

-14.2042 = Validation score (-mean_absolute_error)

60.9s = Training runtime

36.22s = Validation runtime

Fitting model: CatBoost_BAG_L1 ... Training model for up to 1070.96s of the 1070.96s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with ParallelLocalFoldFittingStrategy

-15.8748 = Validation score (-mean_absolute_error)

386.36s = Training runtime

0.21s = Validation runtime

Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 876.26s of the 876.26s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with ParallelLocalFoldFittingStrategy

-14.001 = Validation score (-mean_absolute_error)

76.39s = Training runtime

0.94s = Validation runtime

Fitting model: XGBoost_BAG_L1 ... Training model for up to 835.4s of the 835.4s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with ParallelLocalFoldFittingStrategy

```

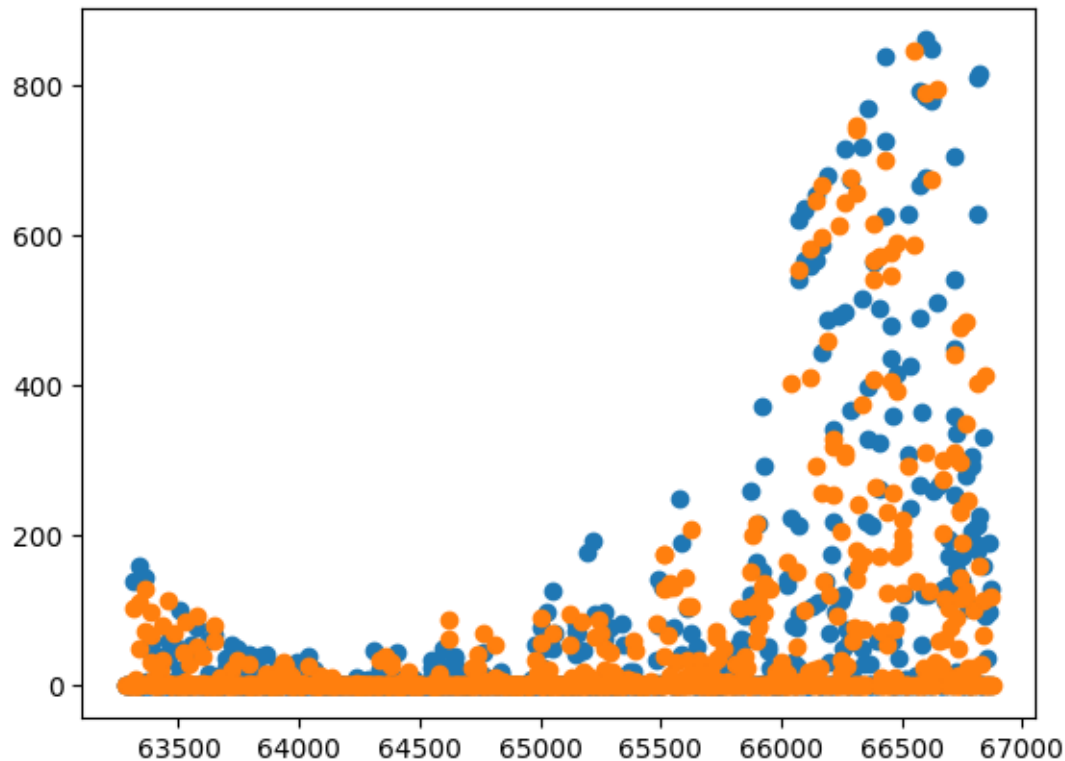
-14.6354          = Validation score    (-mean_absolute_error)
168.32s = Training    runtime
46.84s  = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 744.26s of the
744.26s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -11.6064          = Validation score    (-mean_absolute_error)
    291.11s = Training    runtime
    0.63s   = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 592.71s of the
592.71s of remaining time.
    Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
    -13.3677          = Validation score    (-mean_absolute_error)
    189.73s = Training    runtime
    45.02s  = Validation runtime
Completed 2/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
484.26s of remaining time.
    -11.3969          = Validation score    (-mean_absolute_error)
    0.41s   = Training    runtime
    0.0s    = Validation runtime
AutoGluon training complete, total runtime = 1316.25s ... Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_96_B/")
Evaluation: mean_absolute_error on test data: -13.82015574286899
    Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -13.82015574286899,
    "root_mean_squared_error": -43.61744319903937,
    "mean_squared_error": -1902.4813512214262,
    "r2": 0.9139982536577307,
    "pearsonr": 0.9568816029552053,
    "median_absolute_error": -0.3668253384183716
}

Evaluation on test data:
-13.82015574286899

      model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0    NeuralNetTorch_BAG_L1 -13.779852 -11.606354          0.370063          0.632516
291.110906          0.370063          0.632516          291.110906
1          True          10

```

1	WeightedEnsemble_L2	-13.820156	-11.396950	3.495752	35.908665
351.293035		0.003501		0.000581	0.407487
2	True	12			
2	LightGBMLarge_BAG_L1	-15.759067	-13.367725	7.403612	45.024767
189.730639		7.403612		45.024767	189.730639
1	True	11			
3	NeuralNetFastAI_BAG_L1	-16.618762	-14.000956	0.989892	0.943059
76.389157		0.989892		0.943059	76.389157
1	True	8			
4	LightGBM_BAG_L1	-16.866271	-14.204224	2.615975	36.218588
60.897540		2.615975		36.218588	60.897540
1	True	4			
5	ExtraTreesMSE_BAG_L1	-17.325265	-13.849344	0.521476	1.056313
1.563362		0.521476		1.056313	1.563362
1	True	7			
6	XGBoost_BAG_L1	-17.385425	-14.635356	4.389942	46.841645
168.321279		4.389942		46.841645	168.321279
1	True	9			
7	LightGBMXT_BAG_L1	-17.534210	-14.293414	2.600712	34.219255
58.211280		2.600712		34.219255	58.211280
1	True	3			
8	CatBoost_BAG_L1	-18.010877	-15.874789	0.140189	0.210378
386.362599		0.140189		0.210378	386.362599
1	True	6			
9	RandomForestMSE_BAG_L1	-18.510659	-14.868143	0.485080	1.065722
8.884482		0.485080		1.065722	8.884482
1	True	5			
10	KNeighborsDist_BAG_L1	-30.061916	-23.648941	0.020382	0.410346
0.032739		0.020382		0.410346	0.032739
1	True	2			
11	KNeighborsUnif_BAG_L1	-30.091156	-23.678158	0.017780	0.416663
0.032354		0.017780		0.416663	0.032354
1	True	1			



```
[13]: loc = "C"
      predictors[2] = fit_predictor_for_location(loc)
      leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Beginning AutoGluon training ... Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_96_C/"
AutoGluon Version: 0.8.2
Python Version: 3.10.12
Operating System: Linux
Platform Machine: x86_64
Platform Version: #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail: 205.17 GB / 315.93 GB (64.9%)
Train Data Rows: 24594
Train Data Columns: 34
Tuning Data Rows: 737
Tuning Data Columns: 34
Label Column: y
Preprocessing data ...
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
Label info (max, min, mean, stddev): (999.6, -0.0, 79.8926, 168.407)
If 'regression' is not the correct problem_type, please manually specify
```

the problem_type parameter during predictor init (You may specify problem_type as one of: ['binary', 'multiclass', 'regression'])

Using Feature Generators to preprocess the data ...

Fitting AutoMLPipelineFeatureGenerator...

Training model for location C...

Available Memory: 129886.6 MB

Train Data (Original) Memory Usage: 8.16 MB (0.0% of available memory)

Inferring data type of each feature based on column values. Set

feature_metadata_in to manually specify special dtypes of the features.

Stage 1 Generators:

Fitting AsTypeFeatureGenerator...

Note: Converting 1 features to boolean dtype as they only contain 2 unique values.

Stage 2 Generators:

Fitting FillNaFeatureGenerator...

Stage 3 Generators:

Fitting IdentityFeatureGenerator...

Stage 4 Generators:

Fitting DropUniqueFeatureGenerator...

Stage 5 Generators:

Fitting DropDuplicatesFeatureGenerator...

Useless Original Features (Count: 2): ['elevation:m', 'location']

These features carry no predictive signal and should be manually investigated.

This is typically a feature which has the same value for all rows.

These features do not need to be present at inference time.

Types of features in original data (raw dtype, special dtypes):

('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
...]

('int', []) : 3 | ['is_estimated', 'hour', 'year']

Types of features in processed data (raw dtype, special dtypes):

('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
...]

('int', []) : 2 | ['hour', 'year']

('int', ['bool']) : 1 | ['is_estimated']

0.1s = Fit runtime

32 features in original data used to generate 32 features in processed data.

Train Data (Processed) Memory Usage: 6.31 MB (0.0% of available memory)

Data preprocessing and feature engineering runtime = 0.14s ...

AutoGluon will gauge predictive performance using evaluation metric:

'mean_absolute_error'

This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the metric value.

To change this, specify the `eval_metric` parameter of `Predictor()` `use_bag_holdout=True`, will use `tuning_data` as holdout (will not be used for early stopping).

User-specified model hyperparameters to be fit:

```
{
    'NN_TORCH': {},
    'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}],
    'GBMLarge'],
    'CAT': {},
    'XGB': {},
    'FASTAI': {},
    'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}]},
    'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}]},
    'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
```

Fitting 11 L1 models ...

Fitting model: `KNeighborsUnif_BAG_L1` ... Training model for up to 1799.86s of the 1799.85s of remaining time.

```
-23.6472      = Validation score    (-mean_absolute_error)
0.02s        = Training    runtime
2.42s        = Validation runtime
```

Fitting model: `KNeighborsDist_BAG_L1` ... Training model for up to 1797.2s of the 1797.2s of remaining time.

```
-23.6995      = Validation score    (-mean_absolute_error)
0.02s        = Training    runtime
0.27s        = Validation runtime
```

Fitting model: `LightGBMXT_BAG_L1` ... Training model for up to 1796.85s of the 1796.85s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with `ParallelLocalFoldFittingStrategy`

```
-11.8495      = Validation score    (-mean_absolute_error)
26.77s       = Training    runtime
12.3s        = Validation runtime
```

Fitting model: `LightGBM_BAG_L1` ... Training model for up to 1765.47s of the 1765.47s of remaining time.

Fitting 8 child models (S1F1 - S1F8) | Fitting with `ParallelLocalFoldFittingStrategy`

```
-13.4321      = Validation score    (-mean_absolute_error)
24.04s       = Training    runtime
```



```

5.42s      = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 1738.12s of
the 1738.12s of remaining time.
-16.3363    = Validation score    (-mean_absolute_error)
4.86s      = Training    runtime
0.73s      = Validation runtime
Fitting model: CatBoost_BAG_L1 ... Training model for up to 1731.92s of the
1731.91s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
-13.0045    = Validation score    (-mean_absolute_error)
186.38s    = Training    runtime
0.09s      = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 ... Training model for up to 1544.31s of the
1544.3s of remaining time.
-15.9433    = Validation score    (-mean_absolute_error)
1.02s      = Training    runtime
0.74s      = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 1541.87s of
the 1541.87s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
-14.3155    = Validation score    (-mean_absolute_error)
31.17s     = Training    runtime
0.39s      = Validation runtime
Fitting model: XGBoost_BAG_L1 ... Training model for up to 1509.05s of the
1509.05s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
-13.4046    = Validation score    (-mean_absolute_error)
60.6s      = Training    runtime
5.93s      = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 1444.7s of the
1444.7s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
-13.4402    = Validation score    (-mean_absolute_error)
67.95s     = Training    runtime
0.27s      = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 1375.42s of the
1375.42s of remaining time.
    Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
-12.7665    = Validation score    (-mean_absolute_error)
88.98s     = Training    runtime
11.73s     = Validation runtime
Repeating k-fold bagging: 2/20
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 1277.91s of the

```

1277.91s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-11.764 = Validation score (-mean_absolute_error)
54.47s = Training runtime
26.49s = Validation runtime

Fitting model: LightGBM_BAG_L1 ... Training model for up to 1244.69s of the
1244.69s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-13.2485 = Validation score (-mean_absolute_error)
48.09s = Training runtime
9.61s = Validation runtime

Fitting model: CatBoost_BAG_L1 ... Training model for up to 1216.44s of the
1216.44s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-12.881 = Validation score (-mean_absolute_error)
372.49s = Training runtime
0.18s = Validation runtime

Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 1029.01s of
the 1029.01s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-14.412 = Validation score (-mean_absolute_error)
61.89s = Training runtime
0.8s = Validation runtime

Fitting model: XGBoost_BAG_L1 ... Training model for up to 996.19s of the
996.19s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-13.4943 = Validation score (-mean_absolute_error)
103.47s = Training runtime
8.18s = Validation runtime

Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 948.87s of the
948.87s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-13.5101 = Validation score (-mean_absolute_error)
143.16s = Training runtime
0.65s = Validation runtime

Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 872.07s of the
872.07s of remaining time.

Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy

-12.6783 = Validation score (-mean_absolute_error)
179.54s = Training runtime
21.83s = Validation runtime

Repeating k-fold bagging: 3/20

Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 770.6s of the 770.6s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -11.7186 = Validation score (-mean_absolute_error)

 80.9s = Training runtime

 36.63s = Validation runtime

Fitting model: LightGBM_BAG_L1 ... Training model for up to 737.15s of the 737.15s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -13.214 = Validation score (-mean_absolute_error)

 71.74s = Training runtime

 13.76s = Validation runtime

Fitting model: CatBoost_BAG_L1 ... Training model for up to 708.66s of the 708.66s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -12.7833 = Validation score (-mean_absolute_error)

 559.36s = Training runtime

 0.26s = Validation runtime

Fitting model: NeuralNetFastAI_BAG_L1 ... Training model for up to 520.43s of the 520.43s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -14.457 = Validation score (-mean_absolute_error)

 93.04s = Training runtime

 1.2s = Validation runtime

Fitting model: XGBoost_BAG_L1 ... Training model for up to 486.77s of the 486.77s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -13.4219 = Validation score (-mean_absolute_error)

 147.57s = Training runtime

 11.26s = Validation runtime

Fitting model: NeuralNetTorch_BAG_L1 ... Training model for up to 437.47s of the 437.46s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -13.596 = Validation score (-mean_absolute_error)

 217.55s = Training runtime

 0.96s = Validation runtime

Fitting model: LightGBMLarge_BAG_L1 ... Training model for up to 361.36s of the 361.36s of remaining time.

 Fitting 8 child models (S3F1 - S3F8) | Fitting with
ParallelLocalFoldFittingStrategy

 -12.6342 = Validation score (-mean_absolute_error)

```

269.41s = Training runtime
32.68s = Validation runtime
Completed 3/20 k-fold bagging repeats ...
Fitting model: WeightedEnsemble_L2 ... Training model for up to 360.0s of the
256.86s of remaining time.
-11.3248 = Validation score (-mean_absolute_error)
0.41s = Training runtime
0.0s = Validation runtime

```

AutoGluon training complete, total runtime = 1543.57s ... Best model:

"WeightedEnsemble_L2"

TabularPredictor saved. To load, use: predictor =

TabularPredictor.load("AutogluonModels/submission_96_C/")

Evaluation: mean_absolute_error on test data: -10.943448932340804

Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.

Evaluations on test data:

```

{
  "mean_absolute_error": -10.943448932340804,
  "root_mean_squared_error": -31.11198877171017,
  "mean_squared_error": -967.9558453310196,
  "r2": 0.9230646421934396,
  "pearsonr": 0.9609647483772289,
  "median_absolute_error": -0.6210449779033658
}

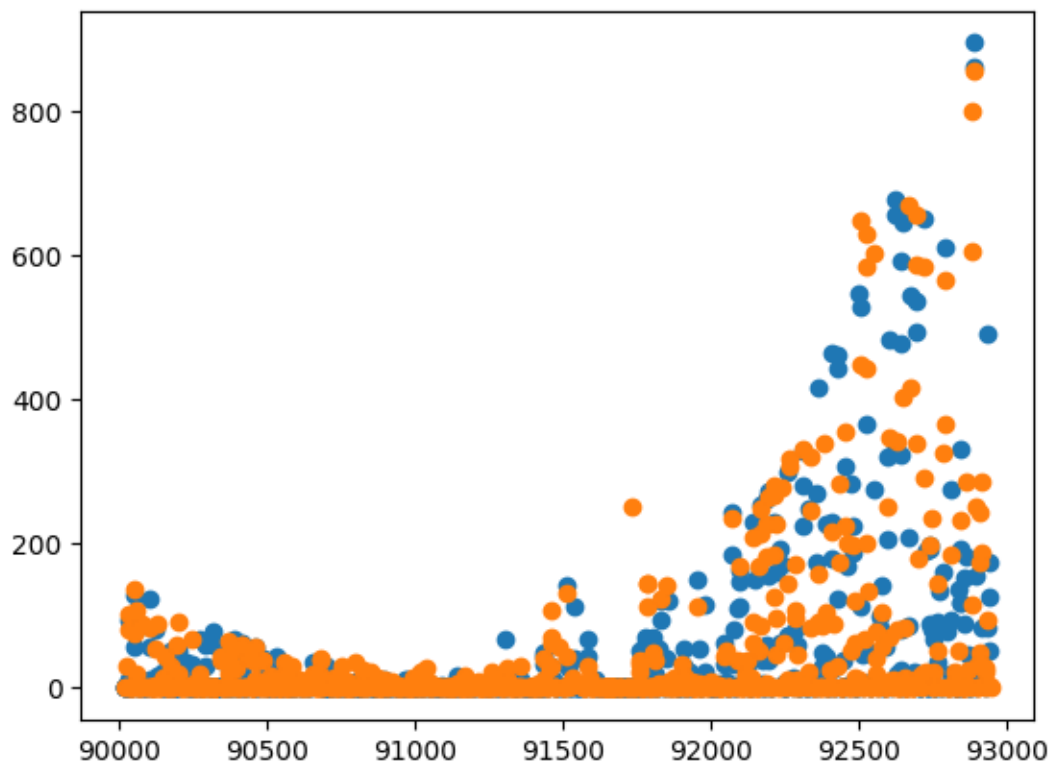
```

Evaluation on test data:

-10.943448932340804

		model	score_test	score_val	pred_time_test	pred_time_val
fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal			
stack_level	can_infer	fit_order				
0	WeightedEnsemble_L2	-10.943449	-11.324826	13.495278	70.277850	
568.270702		0.003744		0.000570	0.409274	
2	True	12				
1	LightGBMXT_BAG_L1	-11.460927	-11.718563	3.735253	36.631079	
80.900352		3.735253		36.631079	80.900352	
1	True	3				
2	LightGBMLarge_BAG_L1	-11.905294	-12.634178	9.238740	32.682299	
269.408155		9.238740		32.682299	269.408155	
1	True	11				
3	LightGBM_BAG_L1	-12.290731	-13.213984	2.306371	13.760183	
71.739522		2.306371		13.760183	71.739522	
1	True	4				
4	NeuralNetTorch_BAG_L1	-12.410340	-13.596046	0.517541	0.963902	
217.552921		0.517541		0.963902	217.552921	
1	True	10				
5	CatBoost_BAG_L1	-12.841863	-12.783341	0.190703	0.262737	
559.360764		0.190703		0.262737	559.360764	
1	True	6				

6	XGBoost_BAG_L1	-12.999348	-13.421861	2.820486	11.259993
147.566018		2.820486		11.259993	147.566018
1	True	9			
7	NeuralNetFastAI_BAG_L1	-14.003227	-14.457019	1.219468	1.195696
93.040765		1.219468		1.195696	93.040765
1	True	8			
8	ExtraTreesMSE_BAG_L1	-15.348385	-15.943328	0.335622	0.744498
1.023689		0.335622		0.744498	1.023689
1	True	7			
9	RandomForestMSE_BAG_L1	-15.973298	-16.336346	0.291116	0.734572
4.864705		0.291116		0.734572	4.864705
1	True	5			
10	KNeighborsDist_BAG_L1	-23.919332	-23.699493	0.013816	0.265059
0.024791		0.013816		0.265059	0.024791
1	True	2			
11	KNeighborsUnif_BAG_L1	-24.102600	-23.647165	0.014048	2.424098
0.024903		0.014048		2.424098	0.024903
1	True	1			



```
[14]: # save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

3 Submit

```
[15]: import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

Loaded data from: X_train_raw.csv | Columns = 36 / 36 | Rows = 92945 -> 92945

Loaded data from: X_test_raw.csv | Columns = 35 / 35 | Rows = 4608 -> 4608

```
[16]: test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",
↪ "location"], left_on=["ds", "location"])

#test_data_merged
```

Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160

```
[17]: # predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

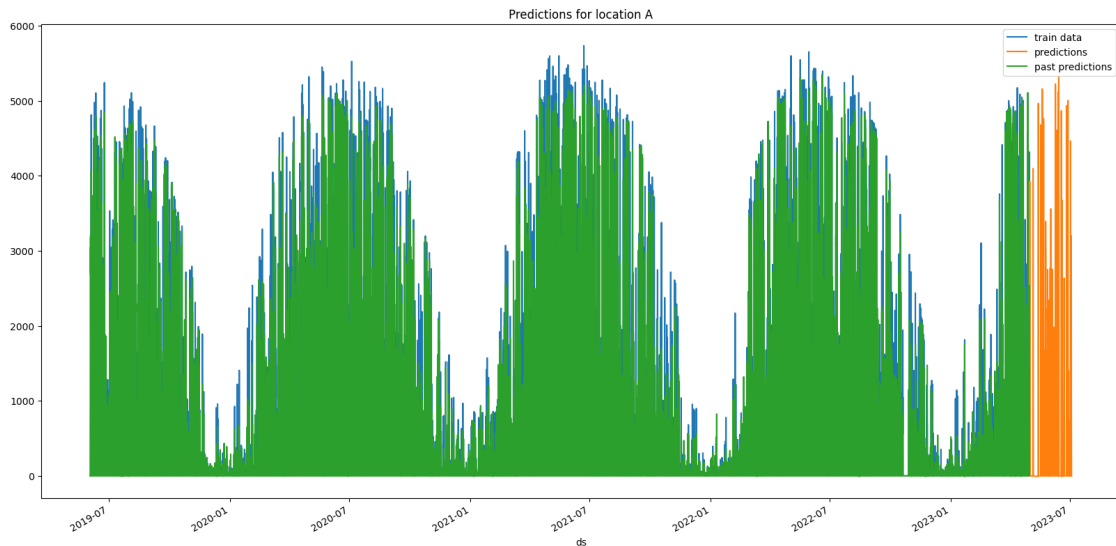
# get past predictions
past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,
↪ "prediction"] = past_pred
```

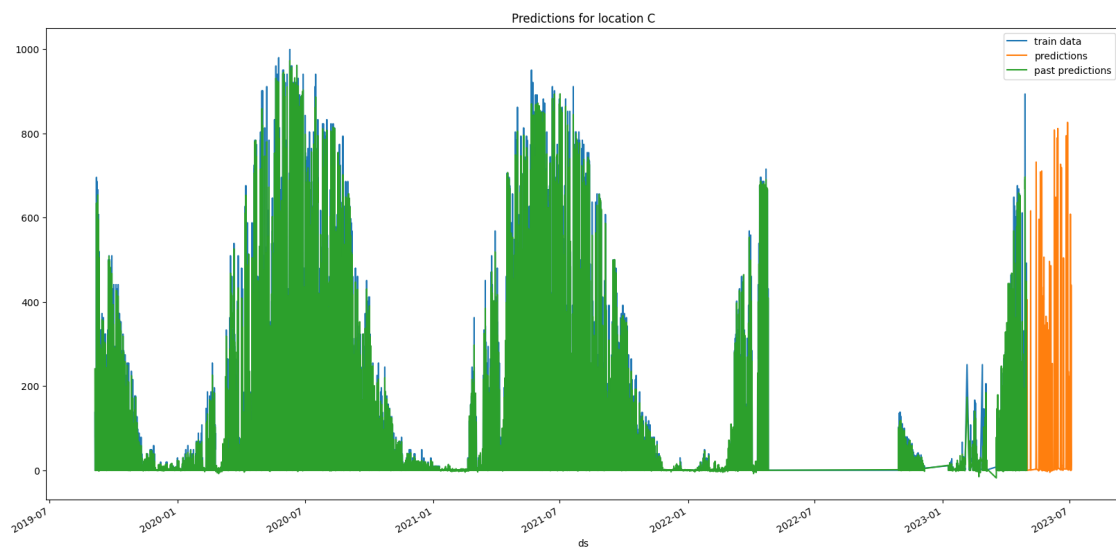
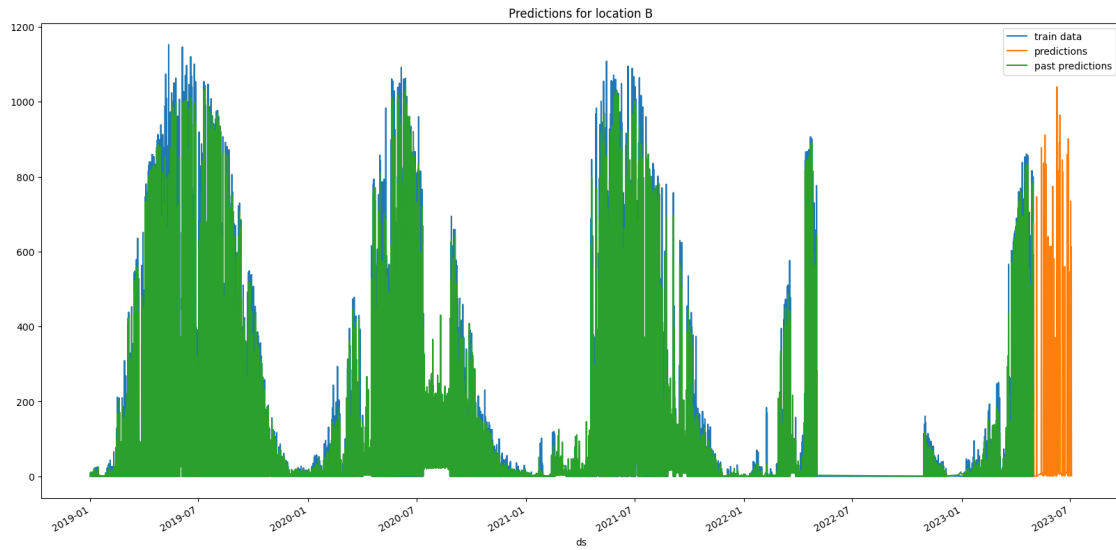
```
[18]: # plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',
    ↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```





```
[19]: # concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
[19]:      id  prediction
0      0   -0.082104
1      1   -0.000746
2      2   -0.273771
3      3   62.498631
```



```

4          4  381.372986
..      ...      ...
715  2155    65.598755
716  2156    41.106895
717  2157     11.369844
718  2158      4.639547
719  2159      1.442424

```

[2160 rows x 2 columns]

```

[20]: # Save the submission DataFrame to submissions folder, create new name based on
      ↪ last submission, format is submission_<last_submission_number + 1>.csv

      # Save the submission
      print(f"Saving submission to submissions/{new_filename}.csv")
      submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
      ↪ index=False)
      print("jall1a")

```

Saving submission to submissions/submission_96.csv
jall1a

```

[21]: # save this running notebook
      from IPython.display import display, Javascript
      import time

      # hei123

      display(Javascript("IPython.notebook.save_checkpoint();"))

      time.sleep(3)

```

<IPython.core.display.Javascript object>

```

[22]: # save this notebook to submissions folder
      import subprocess
      import os
      subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
      ↪ join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
      ↪ ipynb"])

```

[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to <https://pandoc.org/installing.html>.
Continuing with doubts...
check_pandoc_version()
[NbConvertApp] Support files will be in notebook_pdfs/submission_96_files/

```
[NbConvertApp] Making directory
./notebook_pdfs/submission_96_files/notebook_pdfs
[NbConvertApp] Writing 186797 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2051301 bytes to notebook_pdfs/submission_96.pdf
```

```
[22]: CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/submission_96.pdf', 'autogluon_each_location.ipynb'],
returncode=0)
```

```
[23]: # feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,
↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'location', 'prediction']

Computing feature importance via permutation shuffling for 32 features using 4392 rows with 10 shuffle sets... Time limit: 600s...

2039.45s = Expected runtime (203.95s per shuffle set)

507.64s = Actual runtime (Completed 4 of 10 shuffle sets) (Early stopping due to lack of time...)

```
[23]:
```

	importance	stddev	p_value	n \
direct_rad_1h:J	1.590809e+02	2.724940	6.925529e-07	4
clear_sky_energy_1h:J	1.319484e+02	2.762976	1.265023e-06	4
clear_sky_rad:W	1.296194e+02	2.668089	1.201645e-06	4
diffuse_rad_1h:J	1.105292e+02	2.291321	1.227467e-06	4
diffuse_rad:W	9.995892e+01	1.718785	7.005439e-07	4
direct_rad:W	7.166362e+01	2.326761	4.713005e-06	4
sun_elevation:d	6.769284e+01	1.824143	2.695351e-06	4
hour	3.496914e+01	2.064970	2.829283e-05	4
effective_cloud_cover:p	3.495665e+01	2.190691	3.380428e-05	4
sun_azimuth:d	2.821977e+01	1.552842	2.290286e-05	4
is_in_shadow:idx	2.238656e+01	0.852310	7.596526e-06	4
total_cloud_cover:p	1.740795e+01	0.431494	2.097936e-06	4
snow_water:kgm2	1.651445e+01	0.938496	2.522293e-05	4
sfc_pressure:hPa	1.475516e+01	2.170157	4.301319e-04	4
relative_humidity_1000hPa:p	1.169462e+01	0.404922	5.715280e-06	4
msl_pressure:hPa	1.079452e+01	1.642004	4.752177e-04	4

visibility:m	9.974443e+00	0.815795	7.495830e-05	4
is_day:idx	9.968553e+00	0.441618	1.196267e-05	4
wind_speed_10m:ms	9.415063e+00	0.597245	3.505644e-05	4
t_1000hPa:K	9.101788e+00	1.083050	2.293025e-04	4
pressure_100m:hPa	8.841513e+00	0.878574	1.340480e-04	4
fresh_snow_6h:cm	8.682340e+00	0.601390	4.560761e-05	4
cloud_base_agl:m	8.007614e+00	0.683189	8.504063e-05	4
precip_type_5min:idx	7.172939e+00	1.102105	4.895274e-04	4
super_cooled_liquid_water:kgm2	7.148955e+00	0.938225	3.067983e-04	4
ceiling_height_agl:m	7.057437e+00	0.279091	8.512106e-06	4
pressure_50m:hPa	5.699006e+00	0.259217	1.294602e-05	4
snow_depth:cm	4.490204e+00	1.034865	1.609985e-03	4
fresh_snow_3h:cm	3.810029e+00	0.285142	5.748650e-05	4
fresh_snow_1h:cm	2.740081e+00	0.275603	1.389857e-04	4
year	1.369079e+00	0.303183	1.433291e-03	4
is_estimated	-1.042267e-08	0.000000	5.000000e-01	4

	p99_high	p99_low
direct_rad_1h:J	1.670389e+02	1.511228e+02
clear_sky_energy_1h:J	1.400176e+02	1.238793e+02
clear_sky_rad:W	1.374114e+02	1.218274e+02
diffuse_rad_1h:J	1.172209e+02	1.038375e+02
diffuse_rad:W	1.049786e+02	9.493929e+01
direct_rad:W	7.845882e+01	6.486842e+01
sun_elevation:d	7.302017e+01	6.236551e+01
hour	4.099979e+01	2.893848e+01
effective_cloud_cover:p	4.135446e+01	2.855884e+01
sun_azimuth:d	3.275478e+01	2.368477e+01
is_in_shadow:idx	2.487569e+01	1.989743e+01
total_cloud_cover:p	1.866811e+01	1.614779e+01
snow_water:kgm2	1.925528e+01	1.377361e+01
sfc_pressure:hPa	2.109300e+01	8.417311e+00
relative_humidity_1000hPa:p	1.287717e+01	1.051206e+01
msl_pressure:hPa	1.558992e+01	5.999120e+00
visibility:m	1.235694e+01	7.591951e+00
is_day:idx	1.125828e+01	8.678828e+00
wind_speed_10m:ms	1.115929e+01	7.670837e+00
t_1000hPa:K	1.226479e+01	5.938791e+00
pressure_100m:hPa	1.140735e+01	6.275677e+00
fresh_snow_6h:cm	1.043867e+01	6.926007e+00
cloud_base_agl:m	1.000284e+01	6.012391e+00
precip_type_5min:idx	1.039159e+01	3.954291e+00
super_cooled_liquid_water:kgm2	9.889000e+00	4.408911e+00
ceiling_height_agl:m	7.872510e+00	6.242364e+00
pressure_50m:hPa	6.456038e+00	4.941975e+00
snow_depth:cm	7.512480e+00	1.467928e+00
fresh_snow_3h:cm	4.642774e+00	2.977284e+00

```
fresh_snow_1h:cm      3.544965e+00  1.935196e+00
year                 2.254510e+00  4.836481e-01
is_estimated         -1.042267e-08 -1.042267e-08
```

```
[ ]: # feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed,
    ↪time_limit=60*10)
```

These features in provided data are not utilized by the predictor and will be ignored: ['ds', 'elevation:m', 'location', 'prediction']
 Computing feature importance via permutation shuffling for 32 features using 5000 rows with 10 shuffle sets... Time limit: 600s...
 2174.69s = Expected runtime (217.47s per shuffle set)
 468.07s = Actual runtime (Completed 3 of 10 shuffle sets) (Early stopping due to lack of time..)

```
[ ]: display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
    ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),
    ↪"autogluon_each_location.ipynb"])
```

```
[ ]: # import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
    ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
    ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',
    ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is
    ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename
```

```

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b', branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m', commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',
↳ 'origin', branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',
↳ 'origin', branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])

```